Calhoun: The NPS Institutional Archive

DSpace Repository

Theses and Dissertations | 1. Thesis and Dissertation Collection, all items

1988

Channel and switchbox routing using a greedy based channel algorithm with outward scanning technique.

Roderick, Michael J.

Monterey, California. Naval Postgraduate School

http://hdl.handle.net/10945/22939

Downloaded from NPS Archive: Calhoun

http://www.nps.edu/library

Calhoun is the Naval Postgraduate School's public access digital repository for research materials and institutional publications created by the NPS community. Calhoun is named for Professor of Mathematics Guy K. Calhoun, NPS's first appointed -- and published -- scholarly author.

Dudley Knox Library / Naval Postgraduate School
411 Dyer Road / 1 University Circle
Monterey, California USA 93943

FEDCL
C5013 B002

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

R466575

CHANNEL AND SWITCHBOX ROUTING
USING A GREEDY BASED CHANNEL ALGORITHM
WITH OUTWARD SCANNING TECHNIQUE

by

Michael J. Roderick

December 1988

Thesis Advisor Chyan Yang

Approved for public release; distribution is unlimited.

T242301

REPORT DOCUMENTATION PAGE

1a Report Security Classification: Unclassified	1b Restrictive Markings
2a Security Classification Authority	3 Distribution Availability of Report: Approved for public release; distribution is unlimited.
2b Declassification Downgrading Schedule	
4 Performing Organization Report Number(s)	5 Monitoring Organization Report Number(s)
6a Name of Performing Organization: Naval Postgraduate School; 6b Office Symbol (*if applicable*): 62ya	7a Name of Monitoring Organization: Naval Postgraduate School
6c Address (*city, state, and ZIP code*): Monterey, CA 93943-5000	7b Address (*city, state, and ZIP code*): Monterey, CA 93943-5000
8a Name of Funding Sponsoring Organization; 8b Office Symbol (*if applicable*)	9 Procurement Instrument Identification Number
8c Address (*city, state, and ZIP code*)	10 Source of Funding Numbers: Program Element No; Project No; Task No; Work Unit Accession No

11 Title (*include security classification*) CHANNEL AND SWITCHBOX ROUTING USING A GREEDY BASED CHANNEL ALGORITHM WITH OUTWARD SCANNING TECHNIQUE

12 Personal Author(s) Michael J. Roderick

13a Type of Report	13b Time Covered	14 Date of Report (*year, month, day*)	15 Page Count
Master's Thesis	From To	December 1988	149

16 Supplementary Notation The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17 Cosati Codes			18 Subject Terms (*continue on reverse if necessary and identify by block number*)
Field	Group	Subgroup	VLSI ROUTING USING OUTWARD SCANNING TECHNIQUE

19 Abstract (*continue on reverse if necessary and identify by block number*)

The problem of channel routing and for that matter routing in general has been attacked in a heuristic manner vice analytical. This is not necessarily "wrong", however it means that there is not always a solution to the problem. Channel routing is one of the most important phases in VLSI CAD (Very Large Scale Integration Computer Aided Design). It performs the detailed routing of a given channnel. The switchbox is a four sided channel area, rectangular in shape, with nets entering from all four sides. There has been much work done in the channel and switchbox routing areas. The Greedy router, a proven heuristic, is one of the important building blocks for most of today's detailed routers and is used as basis for this thesis. Most routers scan the routing area using a left to right scanning method. This thesis attempts a different variation in routing, using an outward scanning technique. The thesis demonstrates how this new algorithm can be applied to various channel routing problems, by performing tests and making comparisons. The thesis also demonstrates how this new router can be used as a CAD tool. The new router assumes that all pins and wiring lie on a common grid, and that vertical wires are on one layer horizontal on another. The thesis also shows how this new channel router can be modified to allow for a switchbox router implementation.

20 Distribution Availability of Abstract	21 Abstract Security Classification	
☒ unclassified unlimited ☐ same as report ☐ DTIC users	Unclassified	
22a Name of Responsible Individual: Chyan Yang	22b Telephone (*include Area code*): (408) 646-2266	22c Office Symbol: 62Ya

DD FORM 1473,84 MAR — 83 APR edition may be used until exhausted; All other editions are obsolete — security classification of this page

Approved for public release; distribution is unlimited.

Channel and Switchbox Routing Using a Greedy Based Channel Algorithm With Outward Scanning Technique

by

Michael J. Roderick
Captain, United States Marine Corps
B.S., Southeastern Massachusetts University, 1982

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 1988

ABSTRACT

The problem of channel routing and for that matter routing in general has been attacked in a heuristic manner vice analytical. This is not necessarily "wrong", however it means that there is not always a solution to the problem. Channel routing is one of the most important phases in VLSI CAD (Very Large Scale Integration Computer Aided Design). It performs the detailed routing of a given channnel. The switchbox is a four sided channel area, rectangular in shape, with nets entering from all four sides. There has been much work done in the channel and switchbox routing areas. The Greedy router, a proven heuristic, is one of the important building blocks for most of today's detailed routers and is used as basis for this thesis. Most routers scan the routing area using a left to right scanning method. This thesis attempts a different variation in routing, using an outward scanning technique. The thesis demonstrates how this new algorithm can be applied to various channel routing problems, by performing tests and making comparisons. The thesis also demonstrates how this new router can be used as a CAD tool. The new router assumes that all pins and wiring lie on a common grid, and that vertical wires are on one layer, horizontal on another. The thesis also shows how this new channel router can be modified to allow for a switchbox router implementation.

LD
.E51
R6
1985
C.1

TABLE OF CONTENTS

I. INTRODUCTION 1
A. SCOPE AND ORGANIZATION 1
B. ROUTING: THE PROBLEM 1
C. CHANNEL AND SWITCHBOX ROUTING 4

II. BACKGROUND 5
A. GREEDY CHANNEL ALGORITHM AND ROUTING RESULTS 5
B. SWITCHBOX ROUTERS AND ROUTING RESULTS 9
C. ARTIFICIAL INTELLIGENCE APPROACH TO ROUTING 15
D. SILICON COMPILERS AND ROUTING 17

III. ALGORITHM OBJECTIVES AND DEVELOPMENT 19
A. OBJECTIVES 19
B. DEVELOPMENT 20
1. General 20
2. SCMOS/nMOS 21
3. Grid Structure 25
4. Channel Density 26
5. Left Scan Area Top Pin Routing 27
6. Left Scan Area Bottom Pin Routing 32
7. Right Scan Area Top Pin Routing 37
8. Right Scan Area Bottom Pin Routing 38
9. Programming And Source Code 38
10. Similarity And Differences To Greedy 39
11. Input/Output 40
12. Switchbox Routing And Algorithm Extension 40

IV. TESTING AND RESULTS 45
A. GENERAL 45
1. Channel Routing 45
2. Switchbox Routing 45

B. CHANNEL ROUTING COMPARISONS WITH MAGIC 45
C. CHANNEL ROUTING COMPARISONS WITH NPGS 52
D. SWITCHBOX ROUTING COMPARISON WITH MAGIC 56
E. SWITCHBOX ROUTING COMPARISON WITH NPGS 58
F. THE ROUTER AS A CAD TOOL 60

V. CONCLUSION AND DISCUSSION 62

APPENDIX A. NPGS ROUTER USER GUIDE 64

APPENDIX B. C PROGRAM CODES 68

LIST OF REFERENCES 138

INITIAL DISTRIBUTION LIST 140

LIST OF TABLES

Table 1. SOLUTIONS TO BURSTEIN'S SWITCHBOX 11
Table 2. NPGS ROUTER VS THE DETOUR (MAGIC'S) ROUTER EXPERIMENT1 46
Table 3. NPGS ROUTER VS THE DETOUR (MAGIC'S) ROUTER EXPERIMENT2 49
Table 4. NPGS ROUTER VS NPGS ROUTER CHANNEL RESULTS 52
Table 5. NPGS ROUTER VS THE DETOUR (MAGIC'S) ROUTER EXPERIMENT9 57
Table 6. NPGS ROUTER VS NPGS ROUTER SWITCHBOX RESULTS 59

LIST OF FIGURES

Figure 1. VLSI Routing Terminology 3
Figure 2. Greedy's Solution to Deutsch's Channel 8
Figure 3. Detour's solution to Burstein's difficult switchbox 12
Figure 4. Luk's solution to Burstein's difficult switchbox 13
Figure 5. Weaver's solution to Burstein's difficult switchbox 14
Figure 6. Mighty's solution to Burstein's difficult switchbox 15
Figure 7. Artificial Intelligent Routing System 16
Figure 8. Example of Routing by Silicon Compiler 18
Figure 9. Second Level Metal 22
Figure 10. First Level Metal 23
Figure 11. Metal2 contacts 24
Figure 12. Case1 - Pin Entrance From Top of Channel 28
Figure 13. Case2 - Pin Entrance From Top of Channel 29
Figure 14. Case3 - Pin Entrance From Top of Channel 30
Figure 15. Case4 - Pin Entrance From Top of Channel 31
Figure 16. Case1 - Pin Entrance From Bottom of Channel 33
Figure 17. Case2 - Pin Entrance From Bottom of Channel 34
Figure 18. Case3 - Pin Entrance From Bottom of Channel 35
Figure 19. Case4 - Pin Entrance From Bottom of Channel 36
Figure 20. Experiment1 - NPGS solution to Deutsch's difficult channel 47
Figure 21. Experiment1 - Detour's solution to Deutsch's difficult channel 48
Figure 22. Experiment2 - NPGS solution to Burstein's difficult channel 50
Figure 23. Experiment2 - Detour's solution to Burstein's difficult channel 51
Figure 24. Experiment3 - NPGS solution to Burstein's difficult channel 53
Figure 25. Experiment4 - NPGS solution to Burstein's difficult channel 54
Figure 26. Experiment5 - NPGS solution to Burstein's difficult channel 54
Figure 27. Experiment6 - NPGS solution to Deutsch's difficult channel 55
Figure 28. Experiment7 - NPGS solution to Deutsch's difficult channel 55
Figure 29. Experiment8 - NPGS solution to Deutsch's difficult channel 56
Figure 30. Experiment9 - NPGS solution to the 48pin 12net switchbox 57
Figure 31. Experiment9 - Detour's solution to the 48pin 12net switchbox 58

Figure 32. Experiment10 - NPGS solution to the 48pin 12net switchbox 59
Figure 33. NPGS Used As A VLSI CAD Tool 61
Figure 34. Typical User Session Following The Initialize Command 66
Figure 35. Typical User Session Following The Route Command 67

I. INTRODUCTION

A. SCOPE AND ORGANIZATION

The scope of this thesis involves the study of the detailed routing of VLSI cells. The thesis is organized into five chapters. Chapter I introduces the VLSI routing problem and the definitions of channel and switchbox. Chapter II presents background subject matter which is essential for this thesis and also necessary to make clearer to the reader how routing in VLSI is accomplished. Chapter III contains the actual development and implementation of the new routing algorithm. The objectives (for this thesis) are also clearly defined in Chapter III. Chapter IV contains the testing and results of the router. Chapter V is the conclusions.

B. ROUTING: THE PROBLEM

The task of interconnecting a large number of circuit elements in a chip is taking up an exhorbinant amount of design time and chip area. Roughly 30 % of total design time and approximately 60 % of the chip area are utilized just to interconnect the circuit elements [Ref. 1: p. 306]. A tremendous effort is constantly being spent on improving the physical aspects of VLSI design. However, efforts made in improving methods of chip layout can and do account for substantial improvements in overall VLSI circuit design.

The interconnection of circuit elements is known as routing. The VLSI (Very Large Scale Integration) community commonly refers to these circuit elements as "cells". By cells we mean multipliers, adders, shift registers and the like. Figure 1 shows how these cells would be oriented in relation to the routing area. In VLSI routing we are trying to take a group of pins and combine them into individual nets. These pins represent actual grid positions in the routing area. Figure 1 depicts a routing problem which has 38 pins on the top of the channel and 38 pins along the bottom of the channel. Let us refer to "pins" as the set of all pins to be connected. A "net" is a subset of pins. For instance, in Figure 1 we have 2 pins located on the top of the channel with the net number 17. Notice that a net with the number 0 means the pin is not used. The routing problem is to connect each net together via a common wire. The difficulty comes in the fact that nets sometimes cannot be overlapped, routing is usually restricted to a couple of layers and the VLSI chip area is not infinity. Layers refers to the various materials used in the VLSI design process. Some of the layers include metal1, metal2 and

polysilicon to name but a few. "Via" is a term used to define the material which connects multiple layers together. M2contact is used to connect metal1 to metal2. Figure 1 depicts a routing problem and shows some of the terminology, previously mentioned, that is peculiar to VLSI routing.

Figure 1. VLSI Routing Terminology: Channel, pin, net, and cell on silicon real estate.

C. CHANNEL AND SWITCHBOX ROUTING

Routing can be divided into two main categories global or "loose" routing and detailed routing. If a rectangular region of silicon area contains pins on two sides only, then routing tools called channel routers can be used. If the region to be routed contains pins on all four sides, it is known as a switchbox routing problem. The switchbox is a four-sided channel area, rectangular in shape, with nets entering from all four sides. The pins on the sides of these various routing geometries, originate from a particular cell. The difficulty of the channel and switchbox routing solutions is the fact that these routers belong to the categories of NP complete problems [Ref. 2: pp. 463-466]. Consequently, algorithms available today are heuristic in nature. They may fail to find a solution to a routing problem even though a solution may in fact exist.

One of the fundamental algorithms upon which much of routing theory is based is the Lee algorithm [Ref. 3: pp. 346-365]. Lee's algorithm is a classical paper on the grid based routing problem utilizing an expansion wave technique. The expansion wave technique is a method by which cells are routed by expanding their respective pins out (i.e., scanning) along a "wave" front and simultaneously searching for a minimal-distance solution from point A to point B, where points A and B might be pins of two different cells. In addition to finding the shortest path, Lee was also concerned with finding the shortest path which crosses other paths the least amount of times. From Lee's algorithm the global routers were developed. Global routing consists primarily of dividing the entire design routing area into rectangular blocks. These blocks are what comprise the various channel and switchbox type of routing areas. Following the global routers came the more detailed channel and switchbox routers. Much of switchbox routing theory is centered around the "Greedy" channel router, developed by Rivest and Fiduccia [Ref. 4: p. 418]. Because of its importance the Greedy algorithm is presented in the following chapter.

II. BACKGROUND

A. GREEDY CHANNEL ALGORITHM AND ROUTING RESULTS

As mentioned earlier the fundamental building block for many of the channel and switchbox routers is the Greedy channel router. Generally speaking the Greedy algorithm or modifications thereof cannot achieve the optimal solutions [Ref. 5: p. 1], however it is a good starting place. In reality, Greedy based algorithms are easier to program and faster to compute than other alternatives and therefore are presented in many production programs. In particular, both the Detour [Ref. 6: p. 173] and Luk's [Ref. 7: p. 129] switchbox routers are based on the Greedy algorithm. These will be reviewed in detail later. The Greedy algorithm was first proposed by Rivest [Ref. 4: p. 418]. It works by scanning the routing channel from left to right, column by column with one single goal: to minimize the use of horizontal tracks. When applied to Deutsch's "Difficult" channel example, with the channel density of 19, the Rivest's Greedy algorithm can route the channel with 20 tracks. The terms tracks and channel density are defined later in this section. On the other hand, the branch and bound method used by Kernighan [Ref. 8: pp. 50-59] which spent hours of computation time uses 28 tracks. It seems the Greedy algorithm is "the" chosen one of the channel routers. The algorithm is now presented from [Ref. 4: pp. 418-422].

The input for the Greedy router consist of (1) a specification of a channel routing problem and (2) three non-negative integer parameters: initial channel density, minimum jog-length, and the steady-net-constant. The minimum jog length is equal to the channel density divided by 4. Experiments have shown that a higher setting (i.e., a higher jog number) reduces the number of vias and thus produces a better solution, a lower setting tends to reduce the number of tracks used. The router will make no "jogs" shorter than the minimum jog-length. As shown in Figure 2, a jog length setting of 2 produced the best results for the given channel routing problem. The "channel density" of a particular channel routing problem is defined to be the maximum number of nets which have pins on both sides of the line $x = \alpha$ for any α . For example, in Figure 2, the line $\alpha = 13$, contains a channel density of 19. The purpose of the steady-net-constant is to keep to a minimum the amount of times a multi-pin net will change tracks. Typically a value of 10 is assigned to the steady-net-constant. Figure 2 shows the steady-net-constant

net-constant assigned a value of 10. Note also that Figure 2 is the identical routing problem as that of Figure 1.

The Greedy router scans the channel in a left-to-right, column-by-column manner, completing the wiring within a given column before proceeding to the next. The routing area is considered to be a grid composed of vertical lines (columns) and horizontal lines (tracks). Its first step in a column is to make connections to any pins at the top and bottom of the column. The connections are minimal; no more wiring is used than is needed to bring these nets safely into the channel, to the first track which is either empty or contains the desired net.

When routing a given column, the Greedy router classifies each net which has a pin to the right as either rising, falling, or steady. A net is rising if its next pin after the current column will be on top of the channel (say column k) and the net has no pins on the bottom of the channel before column k + steady-net-constant. A net is falling if its next pin after the current column will be on the bottom of the channel (say column k) and the net has no pins on the top of the channel before column k + steady-net-constant. Steady nets are the remaining nets.

The second step in a column tries to free up as many tracks as possible by making vertical connecting jogs that "collapse" nets which currently occupy more than one track. The third step tries to shrink the range of tracks occupied by nets still occupying more than one track, so collapsing these nets later will be less of a problem. Freeing up of tracks has the highest priority, consequently jogs made here have priority over jogs made in the next step.

The fourth step makes "preference" jogs that move a net up if the pin is a rising pin and moves a pin down if the pin is a falling pin. The router chooses longer jogs over shorter ones if there is a conflict. A conflict will occur if a pin is simultaneously classified as both rising and falling. Steps three and four are considered optional steps in the routing algorithm, but making use of them may improve the routing results.

The fifth step is neeeded if a pin could not be connected up in step one because the channel is full. In this situation the router adds a new track to the channel between existing tracks, and connects the pin to this track. When the current column is completely routed, the router extends the wiring into the next column and repeats the same set of procedures.

One nice feature of the Greedy channel router is that its control structure is very flexible and practical. Consequently, it is easy to make changes in the heuristics utilized to achieve special effects or to rearrange priorities. As a result of effectively controlling

various parameters in the routing process such as the number of vias introduced, number of bends, total wire length, the overall performance of the VLSI chip can be improved in terms of speed, capacitance and resistance. In the VLSI "world", silicon real estate is at a premium, therefore designers must use every micron in the most efficient manner possible. The results of the routing of Deutsch's channel by Greedy are shown below in Figure 2.

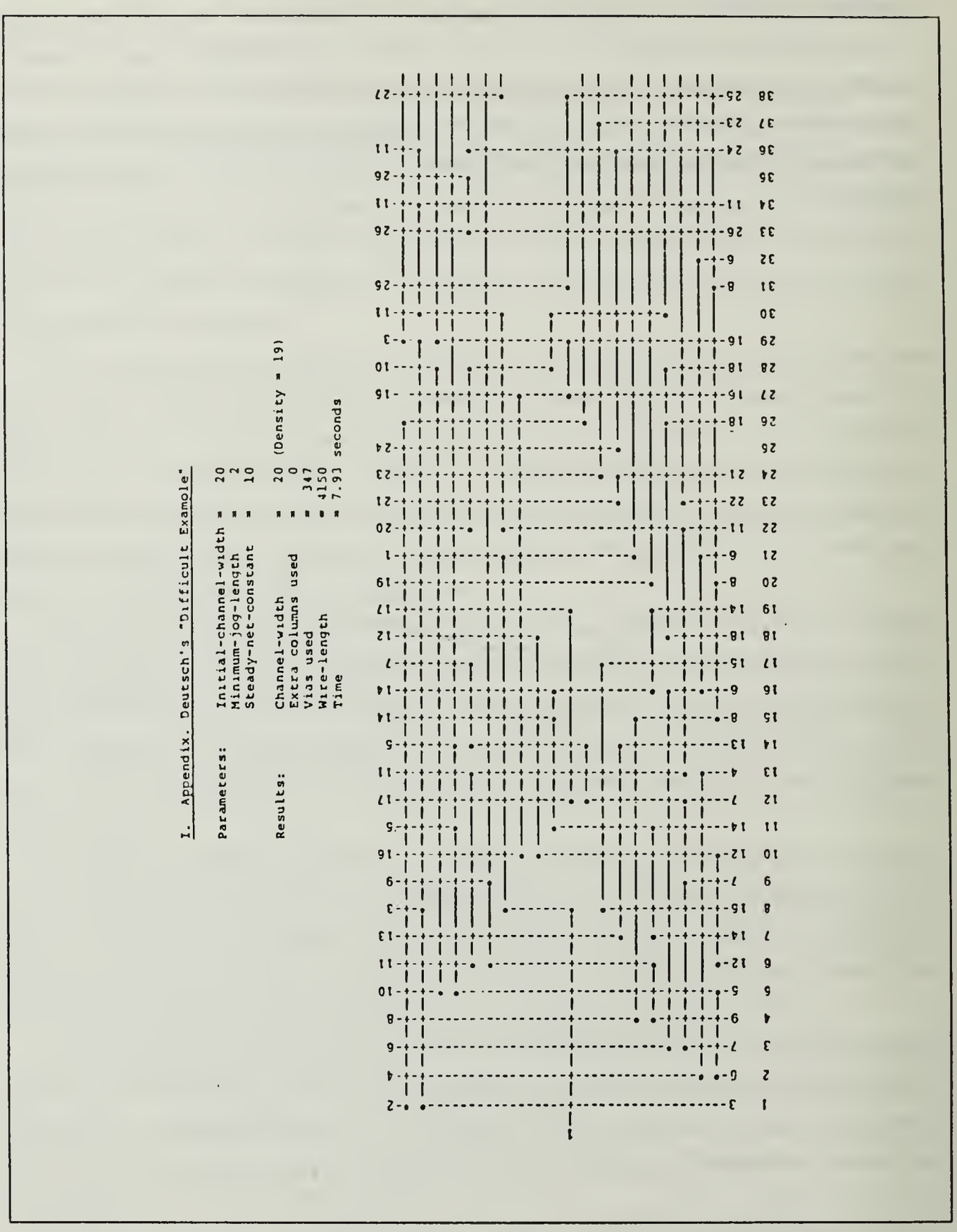

Figure 2. Greedy's Solution to Deutsch's Channel: [Ref. 4: p. 423].

B. SWITCHBOX ROUTERS AND ROUTING RESULTS

The more familiar switchbox routers available include Detour [Ref. 6: pp. 173-179], Mighty [Ref. 9: pp. 2-5], LUK's [Ref. 7: pp. 129-149] adaptation of a greedy channel router, Weaver [Ref. 10: p. 21], and Beaver [Ref. 11: pp. 336-340].

The Detour router is a switchbox router developed as part of the MAGIC layout design system [Ref. 12: p. 1]. It provides routing of switchboxes containing obstacles. This "obstacle avoidance" capability gives designers the option of prewiring special nets such as Vdd, GND, and Clock within the routing area. The router's uniqueness lies in its ability to avoid obstacles within routing regions. Furthermore, Detour allows nets to have an arbitrary numbers of pins on each edge of the channel. In order to route switchboxes and channels successfully, Detour uses two fundamental routing techniques. First, it resolves "cyclical conflicts". Cyclical conflicts occur when a track is needed by two different nets at the same time [Ref. 6: p. 174]. A cyclic conflict may prevent one of the nets from occupying a desired channel track. Consequently, the net must temporarily occupy a second track to avoid conflict with other nets. Detour resolves cyclical conflicts by implementing a strategy that initially tries to move rising and falling nets into tracks which they need to make their own connections and then tries to move them out of tracks needed by other nets to make their respective connections. Second, if a net has more than one pin on the right edge of the channel, the router splits the net to connect these pins.

The Mighty switchbox router is based on an algorithm that routes incrementally and intelligently the nets in the routing region and allows modification by a "rip-up" of nets that may impede the complete routing of other nets [Ref. 9: p. 3]. The unique aspect of Mighty is in the connection or changing of connections already made for nets in the routing region to allow "poor" quality connections to find a better solution. The Mighty router essentially has two basic techniques for making these changes. First, it initiates a "weak" modification. This modification merely pushes aside blocking connections and makes room for the new connections. The second type is known as a "strong" modification. With the strong modification Mighty actually "rips up" the entire process and reroutes the old connections as well as the new connections. The Mighty router begins to approach the realm of artificial intelligent routers, which introduces a fairly new switchbox router called Weaver.

Weaver uses an artificial intelligence approach to solve VLSI routing problems. Artificial intelligence in relation to VLSI routing will be discussed briefly in the following section. The Weaver method is based on knowledge expert sytems and is very complicated. The Weaver method is capable of deciding how and when a net should be routed based upon a wide range of criteria. Weaver is then able to utilize the most optimum method of routing . Specifically, the problem of rising nets, falling nets and jog distance is handled differently in the Weaver method. Weaver employs the rectilinear Steiner tree [Ref. 10: pp. 26-27] to find the best routing pattern for a net and then applies the wire length concept to each segment of the net appropriately. The rectilinear Steiner tree is essentially an algorithm for finding the shortest path between pins of a net. In the past the rectilinear Steiner tree has been used on printed circuit board routing. The wire length concept essentially is a method of ordering nets based on their total wire length.

The Beaver method is a relatively new switchbox router. Beaver's important features are its use of priority queues to determine the order in which nets are routed and its prioritized control of individual track and column usage. Beaver uses a combination of 3 methods to accomplish the routing of nets: a corner router, a line sweep router, and a thread router. Corner routing is done first, line sweep routing second and thread routing is done last. The corner router attempts to connect terminals that form a corner connection. The line sweep router uses the computational geometry technique with the same name as shown in [Ref. 13: pp. 643-647]. This technique joins subnets, where these subnets belong to the same net but require a connection by a common wire. The thread router is a maze-type of router which does not restrict its connection search to any particular form. All three routers are given a priority queue of nets to route. Beaver has successfully routed all of the classic switchbox problems.

Luk's greedy switchbox router is an extended version of the Greedy channel router by Rivest and Fiduccia. Luk surmises that the switchbox routing problem essentially places two additional unplanned constraints on the greedy router. Assuming that the scan direction is from left to right, the additional constraints are:

1. To match the terminals on the left of the routing region.
2. To match the terminals on the right of the routing region.

Luk determined the following:

> To overcome these constraints the router does two additional steps. First the left edge terminals enter directly into the routing region as horizontal tracks. Second, instead of jogging to the next top and bottom terminals as in the fourth step of the

greedy router, the horizontal tracks are jogged to a target row. This target row is a row where a right edge terminal is located [Ref. 7: p. 6].

When comparing the quality of different routers a number of factors may or may not be considered depending on one's desired routing goals. These factors and their affects are listed below:

1. 100 % routing - how much hand routing will be needed.
2. Number of vias - affects capacitance and propagation delays.
3. Total wire length - affects propagation delays and routing area.
4. Number of bends - same as vias.

A classical problem is the Burstein's difficult switchbox [Ref. 14: p. 223] example. The routing results for some of the switchbox routers that have been previously mentioned are listed in Figures 3 through 6, and Table 1. In the via category Mighty generated close to half the amount of vias which Detour produced. In the wire length category Weaver showed the best performance. The Detour router by itself was unable to completely route the switchbox [Ref. 6: p. 178]. However, the makers of Detour were able to find a solution by hand. In order to give the router a "hint", the author's of Detour routed one critical net by hand and then ran Detour again. This time the router was able to complete the switchbox while working around the "hand" routing. The darkened in portion of Figure 3 represents the part that had to be hand routed.

Table 1. SOLUTIONS TO BURSTEIN'S SWITCHBOX

ROUTER	NO.of ROWS	NO.of COLS	NO.of VIAS	WIRE LENGTH
Detour	15	23	67	564
Luk	16	23	58	577
Weaver	15	23	41	531
Mighty	15	22	39	541

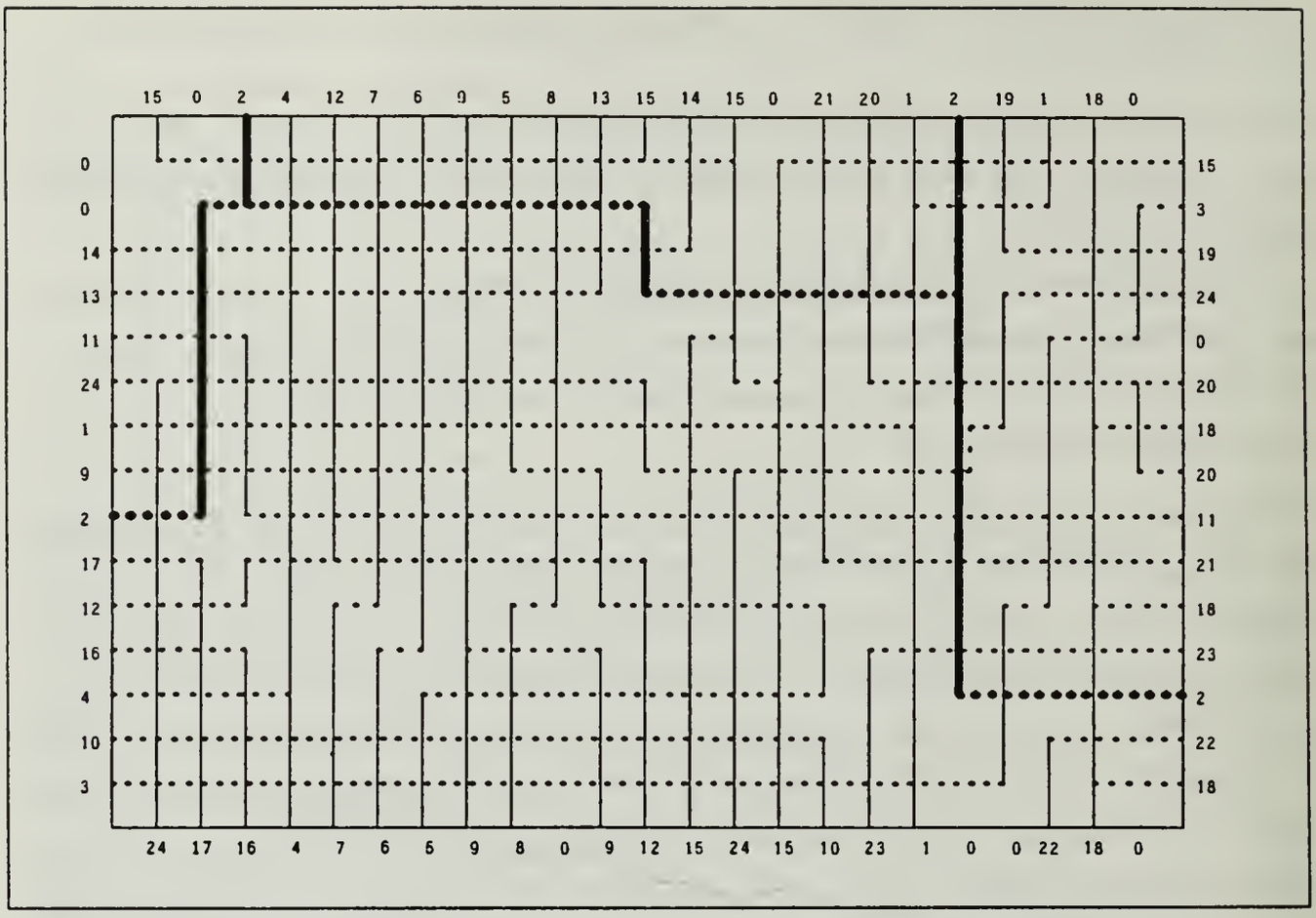

Figure 3. Detour's solution to Burstein's difficult switchbox: [Ref. 10: p. 115].

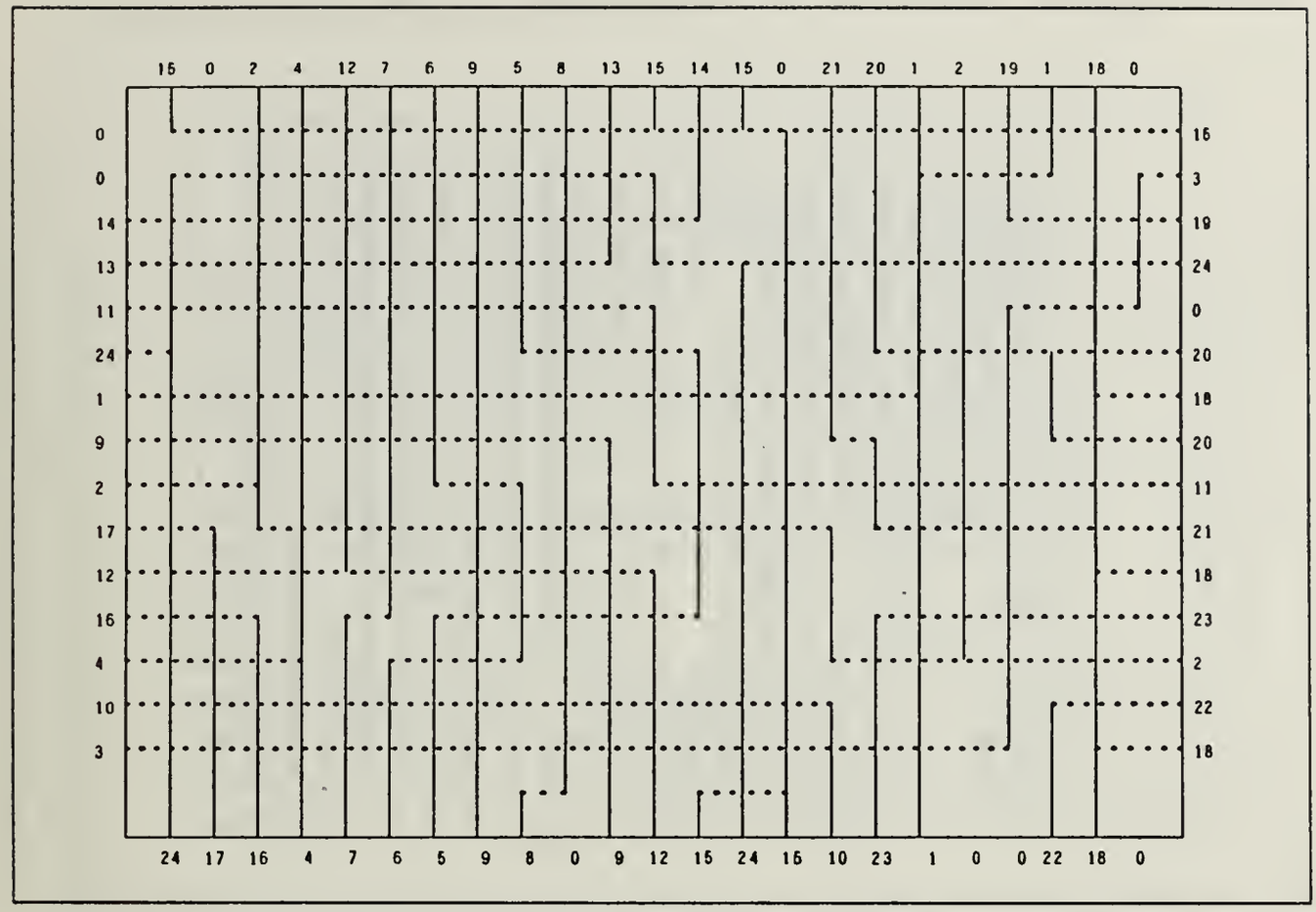

Figure 4. Luk's solution to Burstein's difficult switchbox: [Ref. 10: p. 116].

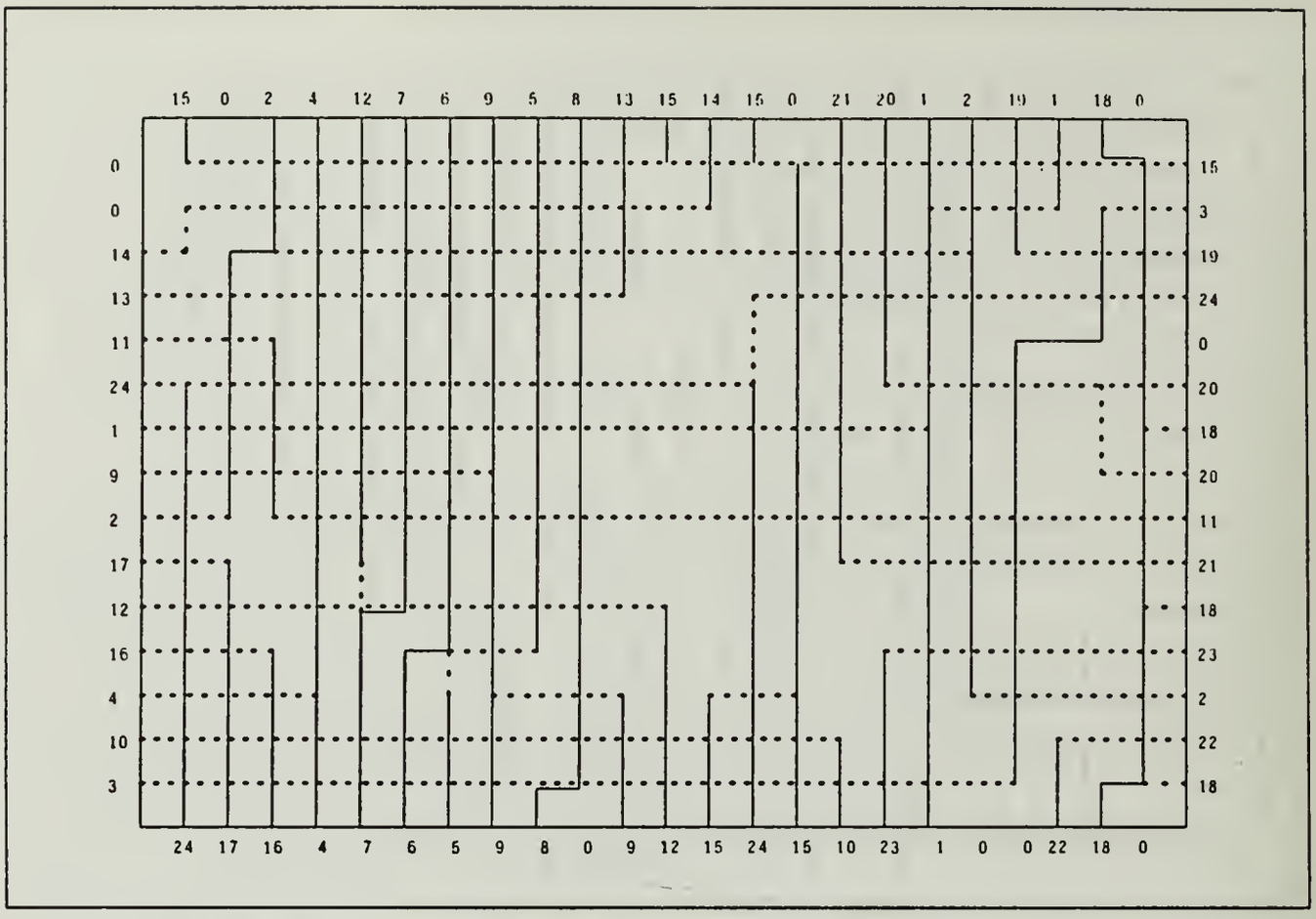

Figure 5. Weaver's solution to Burstein's difficult switchbox: [Ref. 10: p. 119].

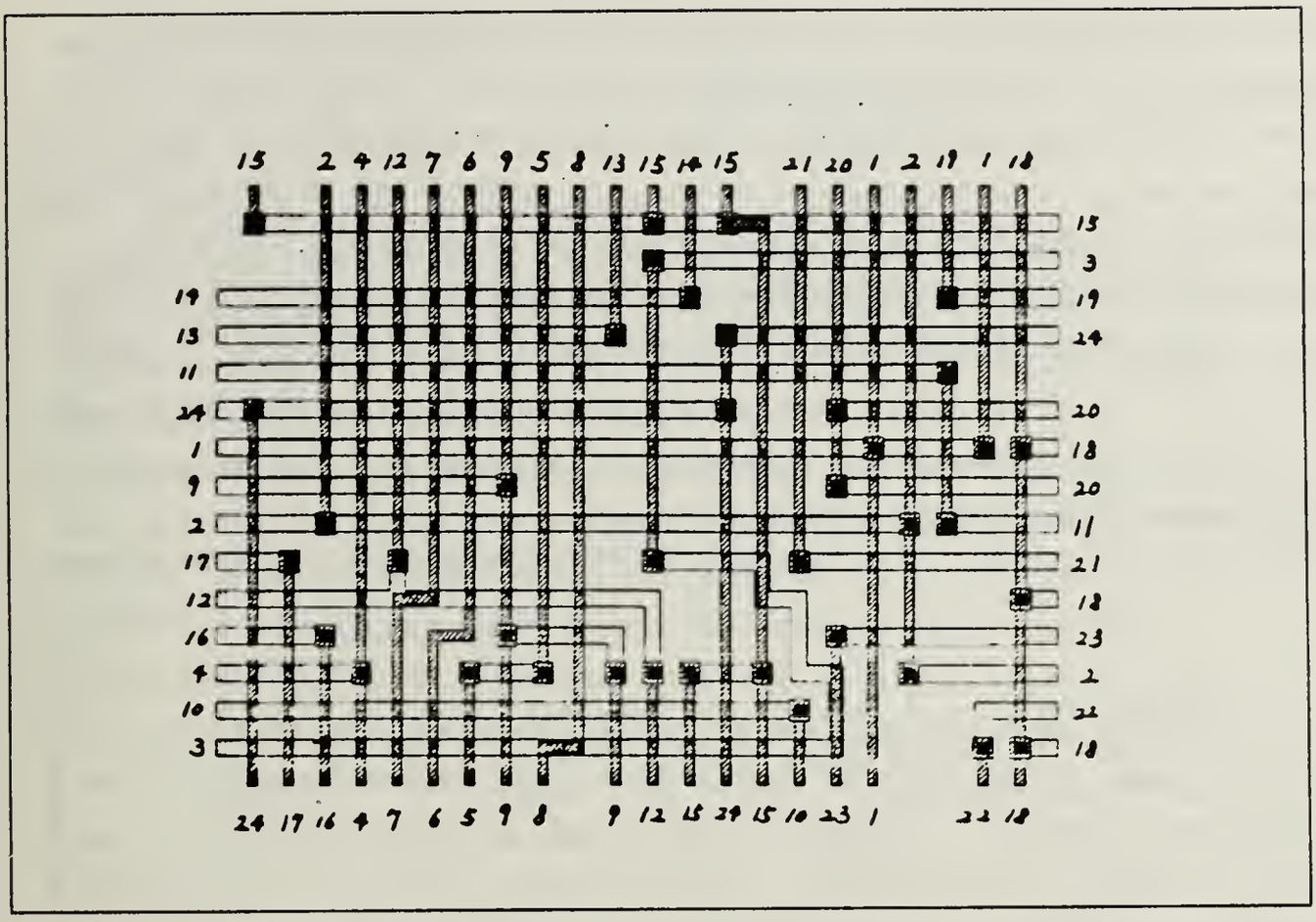

Figure 6. Mighty's solution to Burstein's difficult switchbox: [Ref. 9: p. 5].

C. ARTIFICIAL INTELLIGENCE APPROACH TO ROUTING

Most recently there has been a new development in routing using artificial intelligence. That is, the router performs routing close to what human designers would do. These new routers rely immensely on the knowledge of human expertise in this area. Artificial intelligent (AI) routers do not impose unnecessary constraints. The router considers all the different factors that affect the routing quality and most importantly it allows constant user interaction throughout the routing process. The Mighty [Ref. 9: p. 1] switchbox router has a considerable amount of AI features [Ref. 10: p. 1]. The Beaver [Ref. 11: p. 1] and Weaver [Ref. 10: p. 1] switchbox routers are pure AI type routers.

AI or knowledge-based expert systems are the general class of pattern directed systems [Ref. 10: p. 45]. Pattern directed inference systems have three components: a collection of programs called pattern-directed modules, a data storage, and an executive. The important difference between algorithmic systems and AI is that in the former the

control and execution sequence is fixed, whereas, in the latter the modules to be executed are selected by the executive based on the pattern of data in the data storage. The sequence in which modules are executed is unpredictable. Essentially an AI system has three components: a working memory, a production memory, and an interpreter. The working memory holds data which represents the state of the problem. The production memory asks questions based on criteria and then takes actions. The results of these actions might be the creation, deletion, or modification of one or more working memory modules. The interpreter decides, based on the working memory elements, which rules are eligible for execution and then chooses one rule, based on some predefined strategy, to be executed. Figure 7 shows a typical AI system.

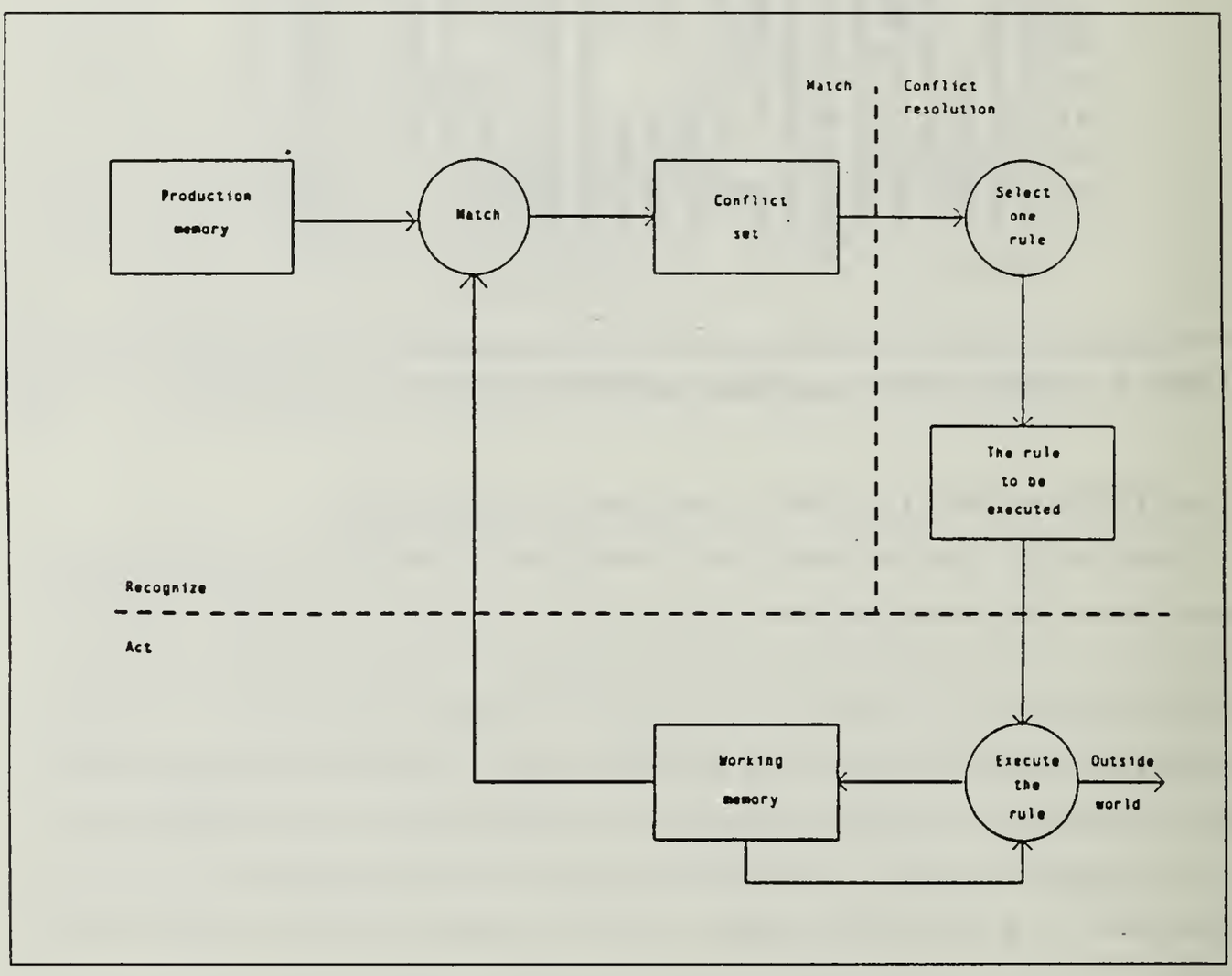

Figure 7. Artificial Intelligent Routing System: [Ref. 10: p. 49].

Presently, AI systems are very costly to implement. This is due mainly to: the lack of knowledge engineers and adequate sophisticated support tools, unfamiliarity of knowledge engineers with the application area, and unfamiliarity of the experts in the application area of AI systems.

Algorithmic approaches to routing are usually very efficient in terms of CPU time, but poor with regards to routing performance. Knowledge-based expert systems or AI systems are inefficient time wise but have good performance [Ref. 10: p. 144].

D. SILICON COMPILERS AND ROUTING

Many of the silicon compilers available today (i.e., Genesil, Mosaico) [Ref. 15: p. 182] , partition the routing area into small regions. Once the regions are determined, they are then ordered so that each region can be routed and its width can be adjusted independently of all previously routed regions. These routing regions are referred to as slicing structures that may have k-bends [Ref. 15: p. 182]. A divide and conquer strategy is then used. The critical differences compared to the standard algorithms are in the method used to determine the next slice (i.e., the process of global routing). Most compilers have an algorithm such that the slicing procedure is repeated until the chip is completely subdivided (i.e., at each step find the minimal-cost slice through the graph). The cost of a slice is determined by the number of bends in the slice, the number of orthogonal edges on the slice, and the number of external junctions that are created by the slice [Ref. 15: p. 183]. Since most users of silicon compilers are unfamiliar with many apsects of the VLSI process, they are predominanly interested in a working product. Consequently the primary goal of most silicon compilers is 100% routability (i.e., no hand routing). An example of the Genesil's silicon compilers routing is shown in Figure 8. The problem is that of the Burstein's difficult channel [Ref. 16: p. 637].

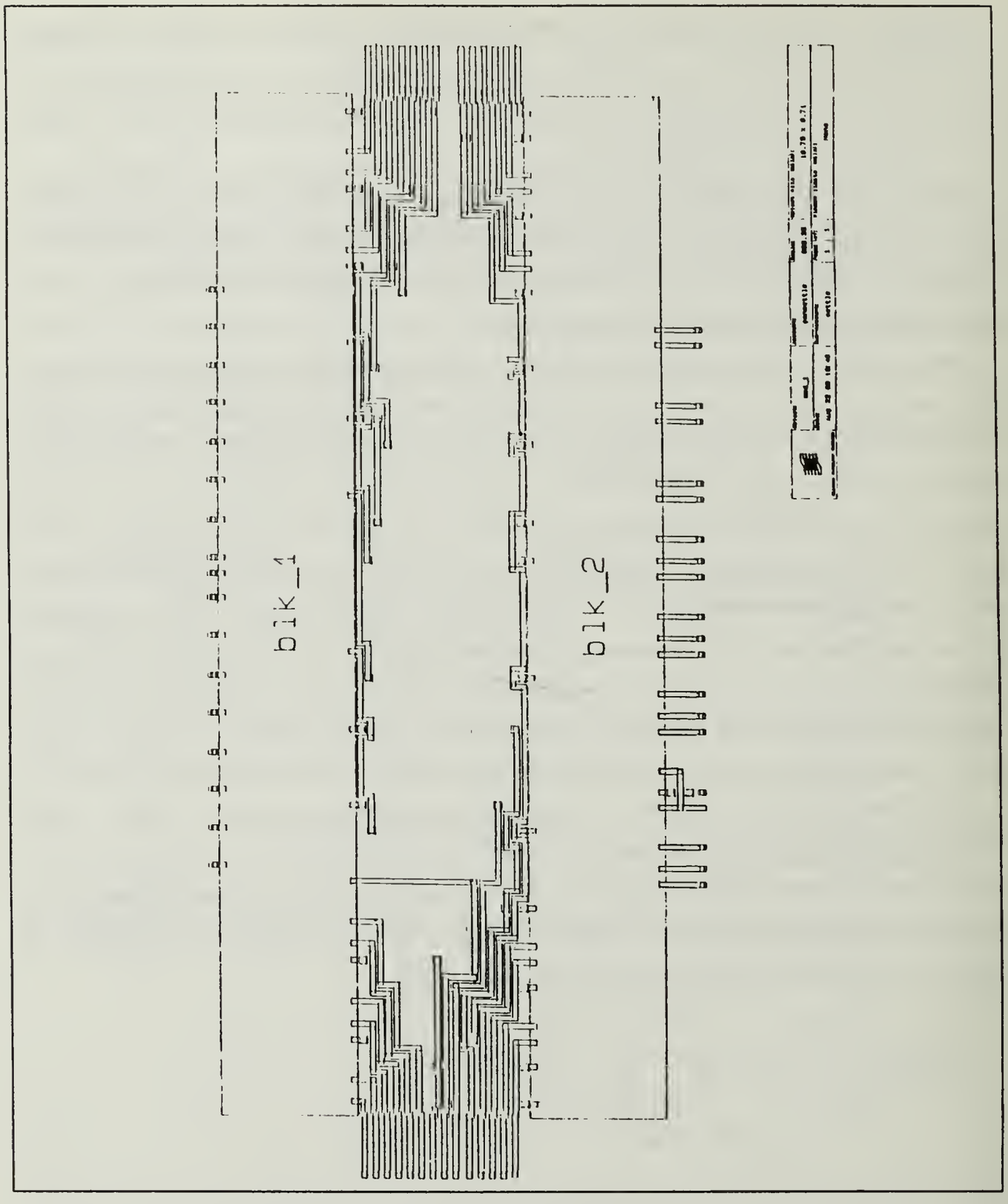

Figure 8. Example of Routing by Silicon Compiler: Figure provided courtesy of the Genesil Silicon Compiler [Ref. 17: p. 1].

III. ALGORITHM OBJECTIVES AND DEVELOPMENT

A. OBJECTIVES

The fundamental building block for this thesis is the Greedy channel router. The algorithm was presented in detail in Chapter II. In this thesis, the author was not concerned with the routing of cells, but rather the routing of pins. By cells we mean multipliers, adders, shift registers and the like. These cells which comprise a complete functional chip are joined together by routing. The requirement to route cells would require much more software development, and for the purposes of this research it was not necessary. The router is only concerned with what is inside the routing area and not what is outside the routing area. The pins in the router unlike Detour's cannot be arbitrarily spaced. The pins are evenly spaced along all sides of the routing region. This eliminates another variable in the routing process, allows for easier program development, and enables the research to concentrate strictly on routing.

The author's objectives were to use the Greedy channel routing algorithm as a basis for developing a new algorithm, with one of the unique variations being the method of scanning the channel. With this modified scanning technique, it was envisioned that a viable channel router could be implemented and finally lead to a switchbox router implementation. The algorithm is designed to have the flexibiltiy to allow the user to vary the columns at which the scanning starts, thereby allowing the comparing of results for these various start positons. The modified algorithm uses an "outward" scanning approach. The scanning begins with the innermost column which contains the maximum track density. From this point the scan moves left until the leftmost column of the routing region is reached. The router then scans right until the rightmost column is reached, thereby completing the routing process. If there is more than one column containing a maximum density value, then the algorithm starts the scan at the leftmost column with the maximum density. The individual steps in the algorithm will be discussed in the development section. The reader should note again that this algorithm only uses the Greedy algorithm as a basis. Essentially the router is "greedy" in nature, because for every new net which enters the routing region, the router establishes a new track. There are many other similarities to the Greedy algorithm which will be outlined in detail later in this thesis.

This outward scanning method was first eluded to in [Ref. 4: p. 422], where the author stated: "Another variation we have not yet tried is to scan outwards from a column of maximum density instead of using a left-to-right scan." The author has performed various tests through out the construction of this modified greedy channel router, with the ultimate goals of this new router to be capable of routing Deutsch's channel (see Chapter III, page 8) and then to modify this new algorithm to solve the switchbox routing problem. Other goals for this thesis are to achieve 100% routability and to reduce or maintain the number of vias used in comparison with Detour's routing (i.e., the MAGIC system). With 100% routing, this implies no hand routing. Reducing the number of vias is accomplished by minimizing the number of times a net is allowed to change tracks and, when possible, allowing the jog length to be a maximum. By reducing the number of vias, or keeping the number of vias "low", the propagation delays are further lowered by reducing the internal capacitance. The routing capacitance between metal1 and metal2 and vias (i.e., m2contact) can be approximated using a parallel plate model [Ref. 18: p. 131]. The capacitance is given by:

$$C = \frac{\varepsilon}{t} A \tag{1}$$

where A is the area of the plate capacitor (i.e., the via), t is the thickness, and ε is the dielectric constant of the insulating material. The developers of Detour make note of the fact that their router does indeed generate a significantly higher amount of vias than those of other routers [Ref. 6: p. 178]. Additionally, it is hoped that this router may serve as a VLSI design tool for use in the VLSI course here at the school.

B. DEVELOPMENT

1. General

The CFL (coordinate free lap) is a method of routing cells by designating them as "symbols" and then using certain CFL commands within a program structure to accomplish the desired routing of these symbols [Ref. 19: p. 3].

The author had originally proposed to use the existing CFL [Ref. 19: p. 3] subroutines and then program a router (algorithm) to be used as part of the CFL. This was deemed to be too cumbersome to learn someone else's code. Similarily, the MAGIC code is too complicated and instead of modifying it, the decision was made to use MAGIC as a basis for comparing routing results of the new algorithm. The author also tried to obtain the source code for other algorithms in order to enhance the comparison

test study of this research but had no success at this endeavor. Consequently the decision was made to develop an algorithm and subsequently write the code to implement it.

2. SCMOS/nMOS

Before commencing the development phase, it was decided to use MOSIS scalable Complementary Metal Oxide Silicon (abbreviated CMOS) design rules. MOSIS service is a prototyping service offering fast turn around standard cell and full-custom VLSI circuit production at low cost.

There were various reasons for this decision. The first is that the past several years have shown a change in the technology from nMOS (n type transistor) to CMOS. The change has occured because CMOS offers excellent performance at low power, and scales very nicely to small feature size. Therefore, development cost are reduced and portability is enhanced. Secondly, because MAGIC supports this and with MAGIC being the only available graphics system for testing of this type of VLSI routing, SCMOS seemed to be the best way to proceed. Layouts designed using SCMOS (the S is short for scalable) rules may be fabricated using either P-well or N-well technology at a variety of feature sizes [Ref. 12: p. 1, manual#2]. MOSIS currently supports fabrication at both .7 microns/λ and 1.5 microns/λ. Interconnecting layers are metal1, metal2 and vias. For this router the only via utilized is m2contact. The layers and their corresponding design peculiarities are shown in Figures 9 through 11 [Ref. 12: pp 2-4, manual#2]. It should be noted that the words "unit" and λ are used interchangeably. Henceforth, in this thesis, a λ is that unit which contains 1.5 micron of wire length. It should also be mentioned here that the MAGIC system has a design rule checker (DRC). Consequently, when editing or testing the router against a certain routing area, the MAGIC system automatically checks that the design is within MOSIS rules. If the rules are violated, MAGIC will display little white dots in the vicinity of the violation. However, the algorithm is designed so as to conform to the MOSIS rules (see Figures 9 through 11) without the user having to correct for these in any way. The DRC was simply a nice feature to have in the process of developing the router.

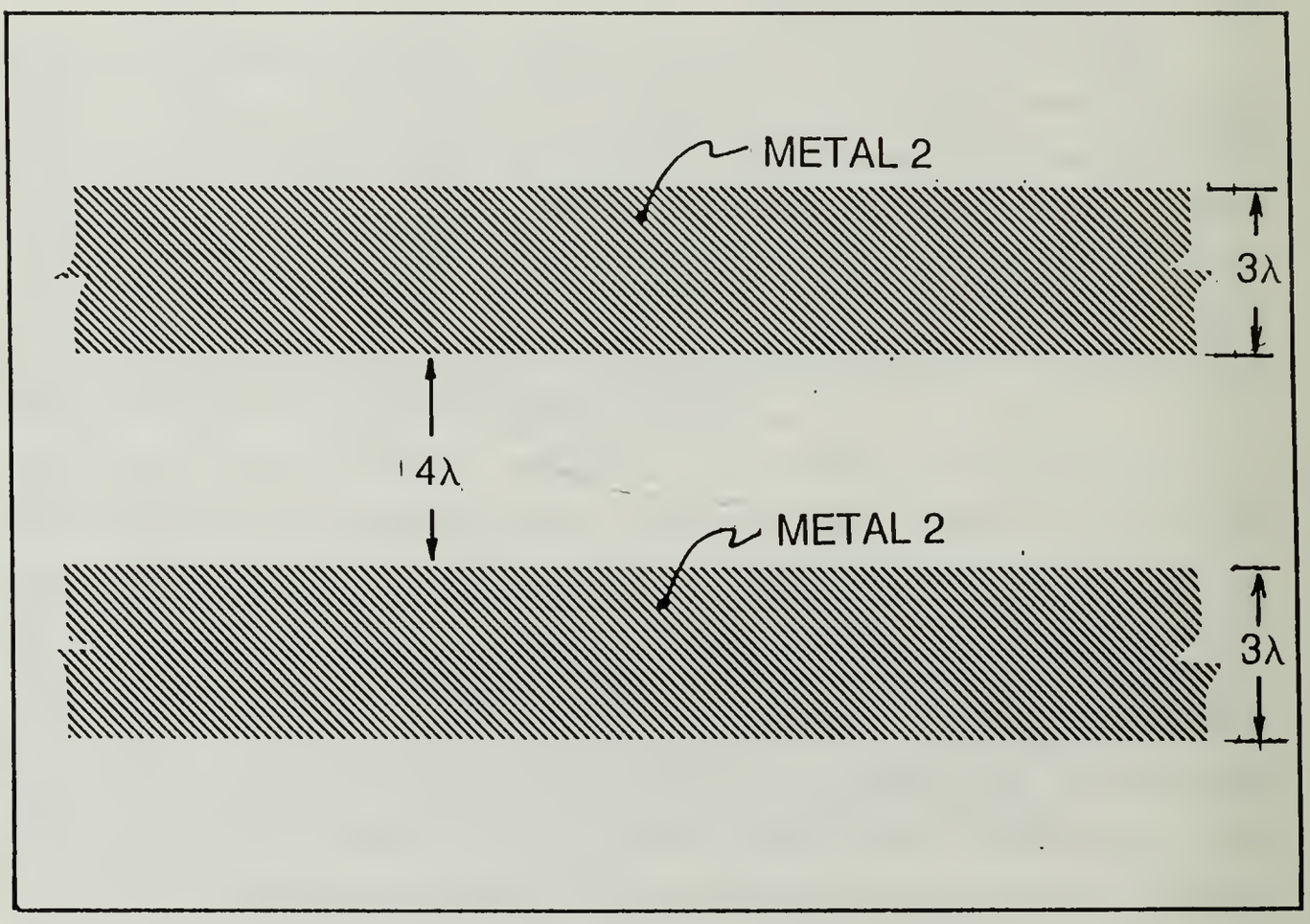

Figure 9. **Second Level Metal:** The top level of this metal is drawn in purple color, and is referred to as metal2. It must always be at least 3λ wide, and metal2 area must be separated from each other by at least 4λ.

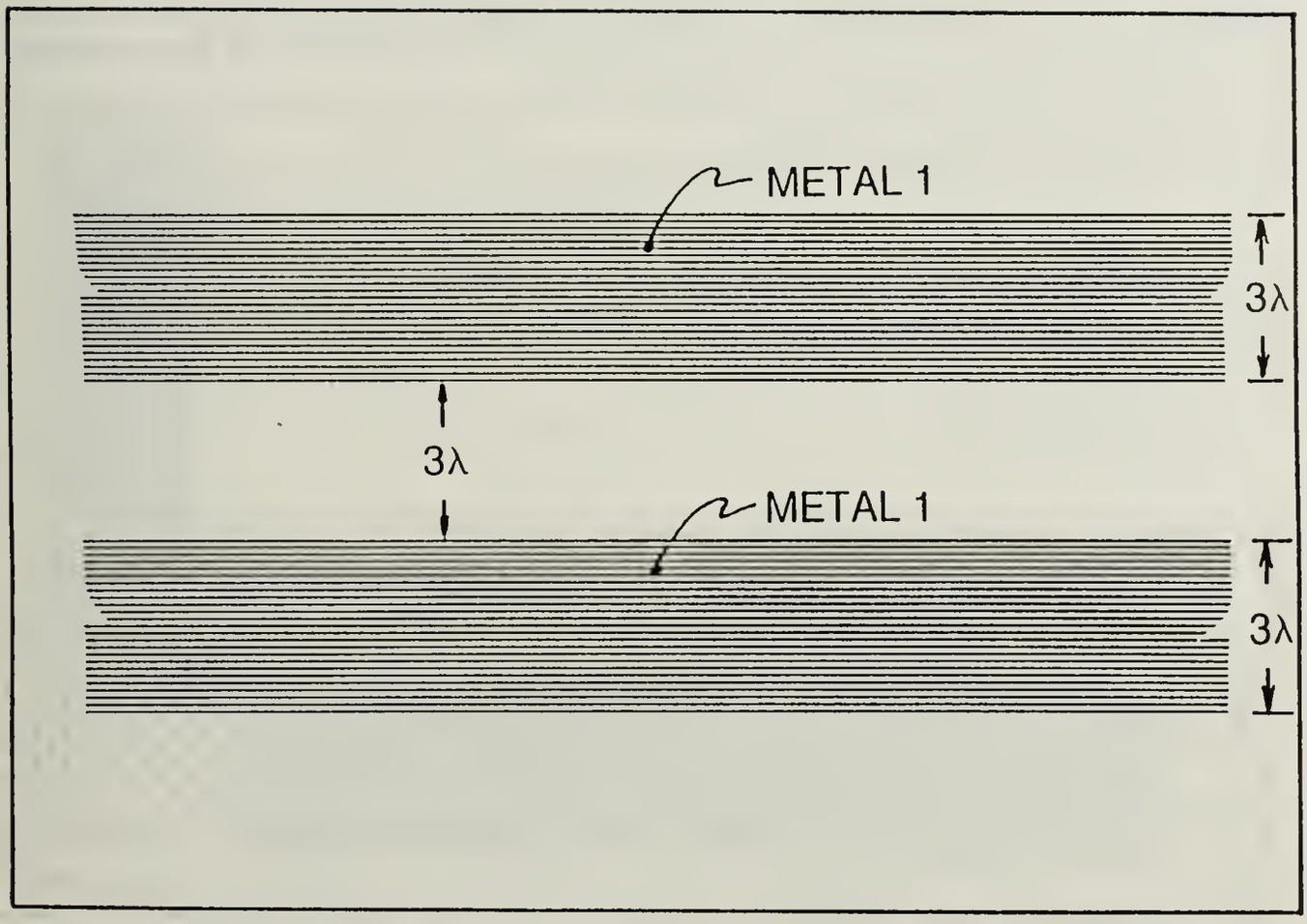

Figure 10. **First Level Metal:** The top level of this metal is drawn in blue and is referred to as metal1. It must always be at least 3λ wide, and metal1 area must be separated from each other by at least 3λ.

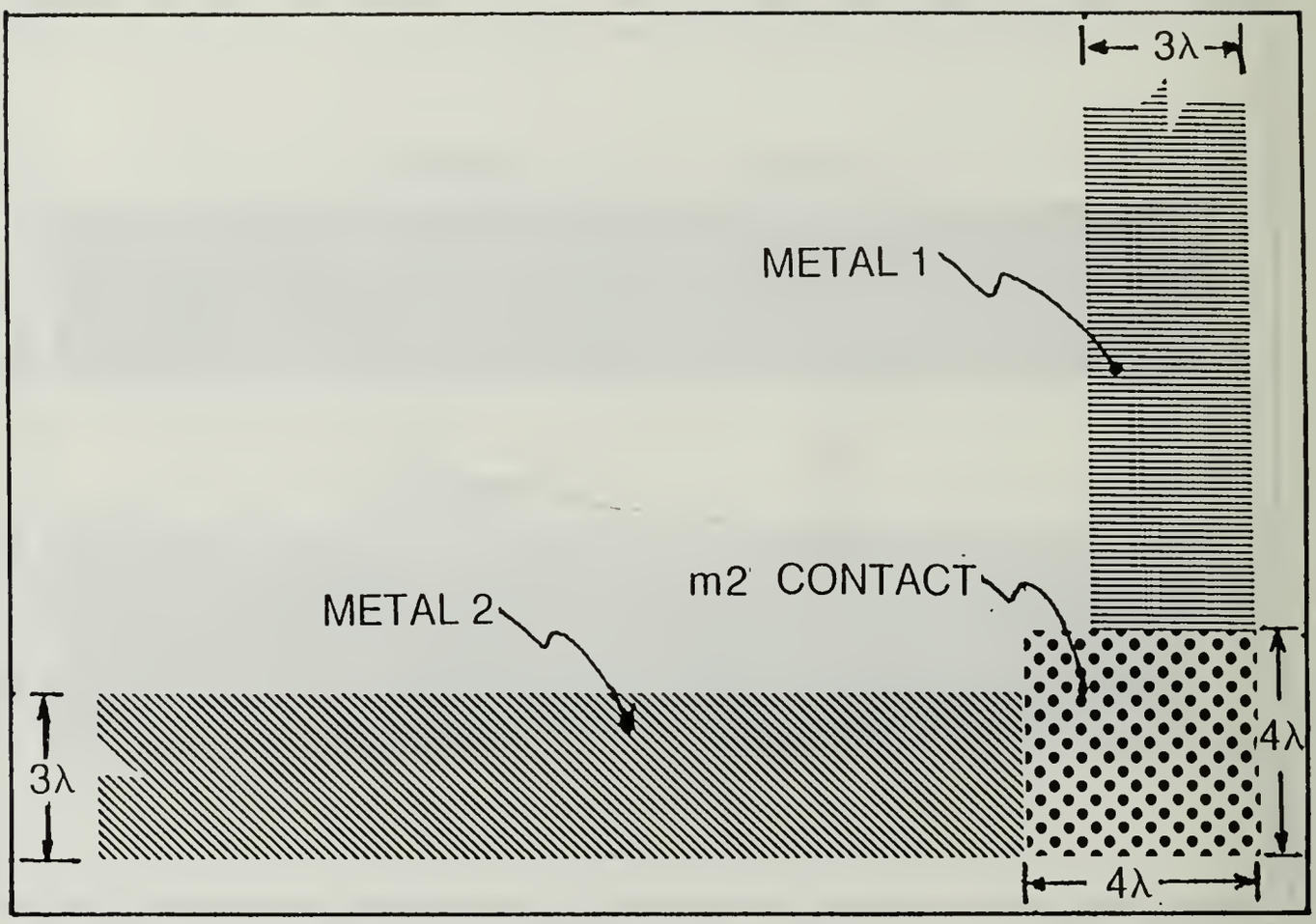

Figure 11. **Metal2 contacts:** Metal2 contact appear as an area with metal1 and metal2 overlapping each other.They must always be at least 4λ wide, and additionally there must not be any polysilicon or diffusion edges underneath the area of the contact or within 1 λ of the contact. Metal2 contacts are also known as m2contacts.

The primary difference as mentioned earlier between nMOS and CMOS is low power consumption in CMOS. The essential difference between SCMOS and nMOS in relation to this research is that in nMOS there is only metal1. There is also no m2contact vias in nMOS. Note, however, Figure 10 is still applicable.

3. Grid Structure

The router utilizes a grid based approach with Manhattan geometry. Grid based means that the routing region is subsequently broken down into rows and columns depending on the channel density and the number of pins to be routed. Manhattan geometry means that only horizontal and vertical lines are allowed. Metal2 is used in the vertical direction and metal1 in the horizontal direction. There are some routers which do allow for routing in a diagonal direction.

The size of the various routing configurations and connections were predicated by the DRC (see section 2). By use of the DRC and countless observations of routing problems, we were able to produce a set of "primitive" map region configurations (see Figures 12-19). These primitives helped to pave the way for further algorithm development.

The program code uses the C language and defines a structure (multiple data storage area) called the *MAP_REG*, i.e., the map region. The map region divides the channel area up into individual grids. The number of grids is based on the rows determined by the channel density routine which shall be discussed later and on the number of columns set forth by the number of pins on the top and bottom of the routing area. Hence the number of grids is equivalent to the number of rows multiplied by the number of columns. Each grid is $K\lambda$ in height and 16λ wide. When routing a channel, K is valid for $K \in [7\lambda, \infty]$. This feature allows the user to obtain the best possible solution and still conform to the MOSIS rules. If all the pins entering the channel are metal2, then the value of K could be set at 7λ. However, if the pins surrounding the routing region consist of metal1, then the value of K would have to be set at a minimum of 11λ. If the router is routing a switchbox, the value of K would be set at a minimum of 19λ. The criteria for determining the above dimension of 19λ was based on meeting the MOSIS rules requirements for SCMOS and to account for the establishing of more tracks needed beyond the density limit when routing a switchbox.

Surrounding each grid is a 4λ "guard" region along the top and right side of the grid. This 4λ is included in the 16λ total width. The guard region allows the routing to take place anywhere inside the grid and at the same time not violate any MOSIS design rule. The routing is done along the bottom of the grid, along the left side of the grid,

and along the right side of the grid. The router uses no more than one horizontal connector (track) per grid when routing a channel. The pins are extended to the tracks by using vertical connectors. When it does channel routing, the router never uses more than two vertical connectors in each grid. The tracks use the bottom of the grid. The top pins use the left side of the grid, and the bottom pins use the right side of the grid. The routing of the top pins is discussed initially in section 5 and depicted in Figures 12-15. Similarly, the routing of the bottom pins is discussed initially in section 6 and depicted in Figures 16-19. The routing of pins in a switchbox is discussed in detail in section 12 of this Chapter. When routing a switchbox, the router may place an additional track horizontally along the center of each grid as needed. The router may also require more than two vertical connectors in a particular column when routing a switchbox. A complete discussion of the switchbox routing extension of this algorithm is contained in section 12 of this Chapter. Whether the router is routing a channel or a switchbox, it always maintains one track per net per column.

4. Channel Density

The first step in the greedy channel routing process as is the case for this router is to compute the channel density. The channel density was previously defined in the discussion pertaining to the Greedy algorithm (see Chapter II page 4). According to [Ref. 7: p. 4], optimality can be achieved if one can guarantee for each column, there is only one horizontal track for each net. Therefore, the channel density determines the number of rows.

The reader should note that the channel density is the lower bound of channel width needed to complete the routing process. When the algorithm cannot route with the given channel density, it will indicate so. The user can then restart the algorithm with a new channel density value or the expected channel width. The C language code for computing the channel density is listed in Appendix B, in a program called *number.c*.

Once the channel density has been determined, the scanning from the column with maximum channel density can commence scanning in the left direction. If there is more than one column having a maximum density, then the router will start the scanning from the leftmost column with maximum channel density. The router has two inputs, the channel density and the starting column. Once the left scanning is completed the right scanning can begin and is done similar to the left scan with some exceptions, which will be pointed out in the following sections. The scanning in both left and right regions is further broken down into top scan and bottom scan areas.

5. Left Scan Area Top Pin Routing

The next step in the routing process is to enable the top pins in the left scan area (i.e., the area to the left of the column with the maximum channel density) to make their initial entrance into the routing region. This is accomplished by asking three essential questions for each pin at every column in the following order:

1. Is there a track already occupied having the same net number as the pin attempting to make an initial entrance into the routing region?
2. If there is not a track already occupied, with the same net number, is there an empty track?
3. If the answers to both one and two were no, is the routing region fully occupied?

If the answer to question 1 is yes then the pin moves to this track and connects. The reader is encouraged to refer to Figures 12 through 15, to better understand how this is physically accomplished.

If the answer to the first question is no and the second question is yes then the pin moves into the routing region and occupies a new track. The track that is selected is the one closest to the top channel borders and also unoccupied. This is accomplished by using one of the schemes depicted in Figures 12 through 15. The scheme (one of the four) is dependent upon the type of material the pin is and if the adjacent grid track is free. Once the new track is established the algorithm looks "ahead" at all pins in the left scan area and extends this new track to the net's leftmost boundary in the routing region. The leftmost boundary being a column which has either a top pin or bottom pin net number the same as the net number of the pin currently being routed. The algorithm then looks "behind" into the right scan area at both top and bottom pins, and finds the net's rightmost boundary in the routing region. The algorithm then extends the track to this rightmost boundary. The rightmost boundary being a column which has either a top pin or bottom pin net number the same as the net number of the pin currently being routed. At this point in the algorithm the net now occupies a track in the range [a, b]. Where a is the leftmost boundary of the net and b is the rightmost boundary of the net.

If the answers to questions 1 and 2 were no then the algorithm sends a message indicating the channel density has been exceeded and must be increased. The router is now ready to start the routing of the bottom pins in the left scan area.

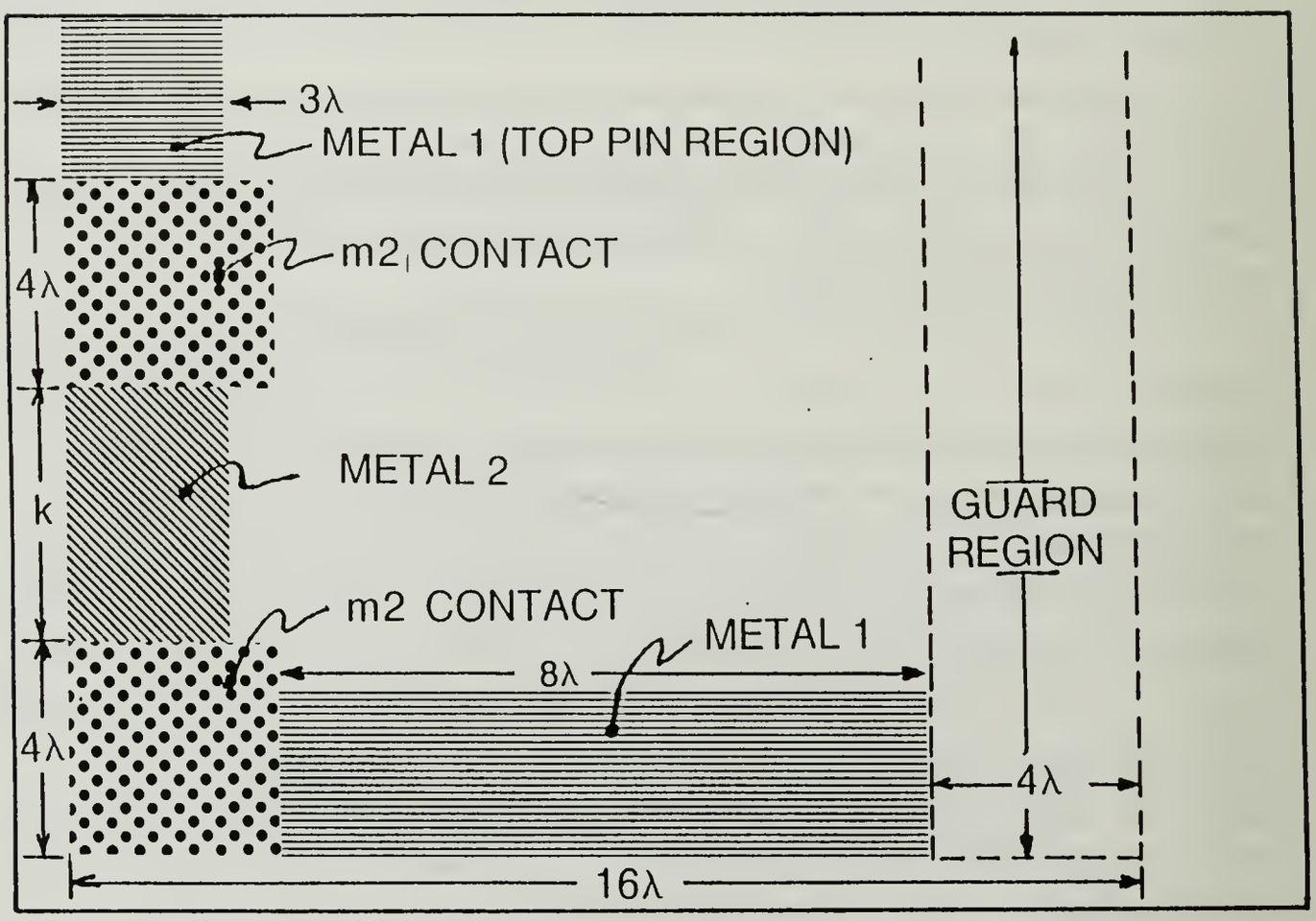

Figure 12. Case1 - Pin Entrance From Top of Channel: Entrance into channel with pin being metal1 and adjacent grid track is equal or unoccupied. In this figure, k is set equal to a minimum of 4λ and has maximum setting which is dependent on the value selected for K (see section 3 page 25).

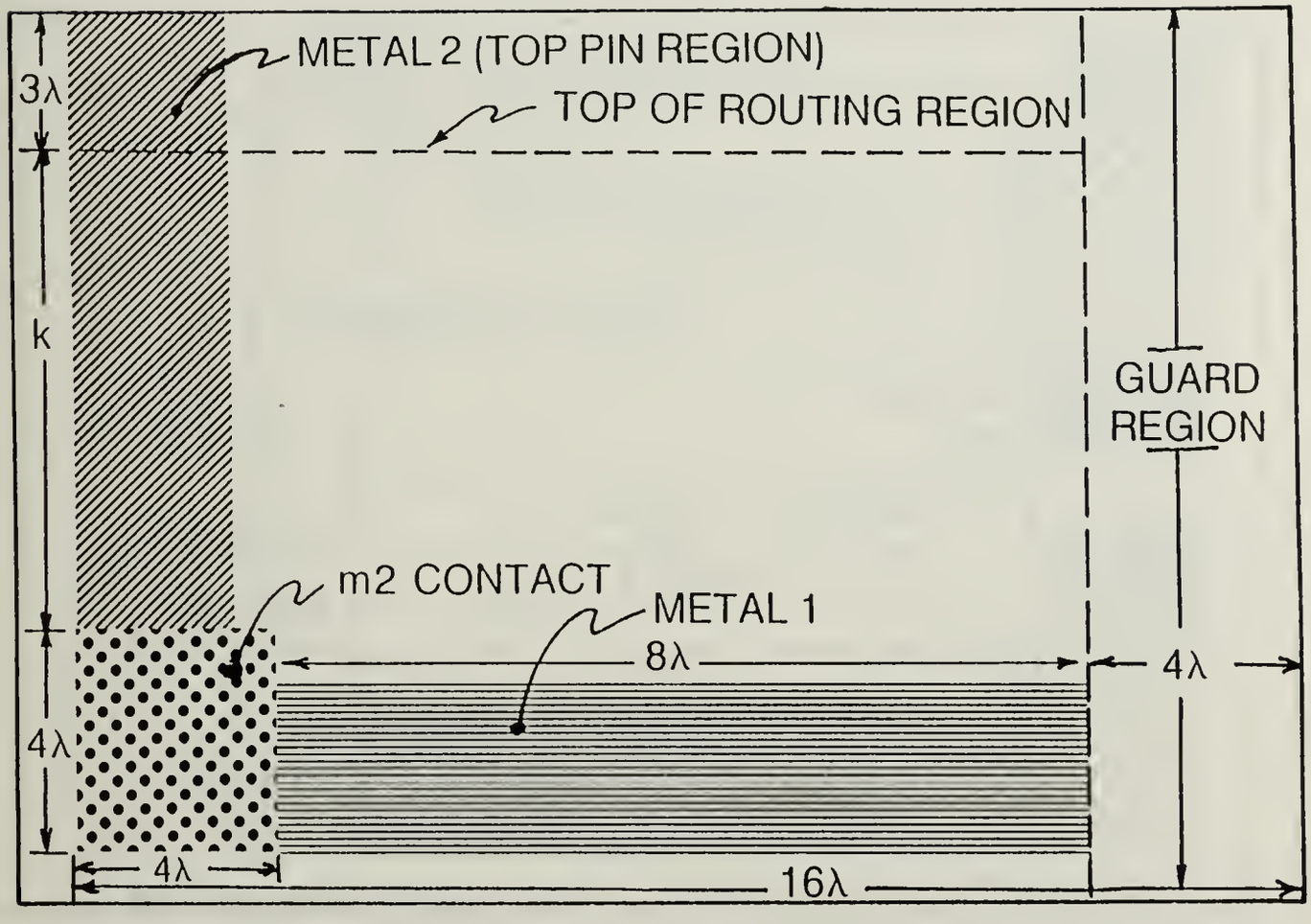

Figure 13. Case2 - Pin Entrance From Top of Channel: Entrance into channel with pin being metal2 and adjacent grid track is equal or unoccupied. In this figure, k is set equal to a minimum of 3λ and has maximum setting which is dependent on the value selected for K (see section 3 page 25).

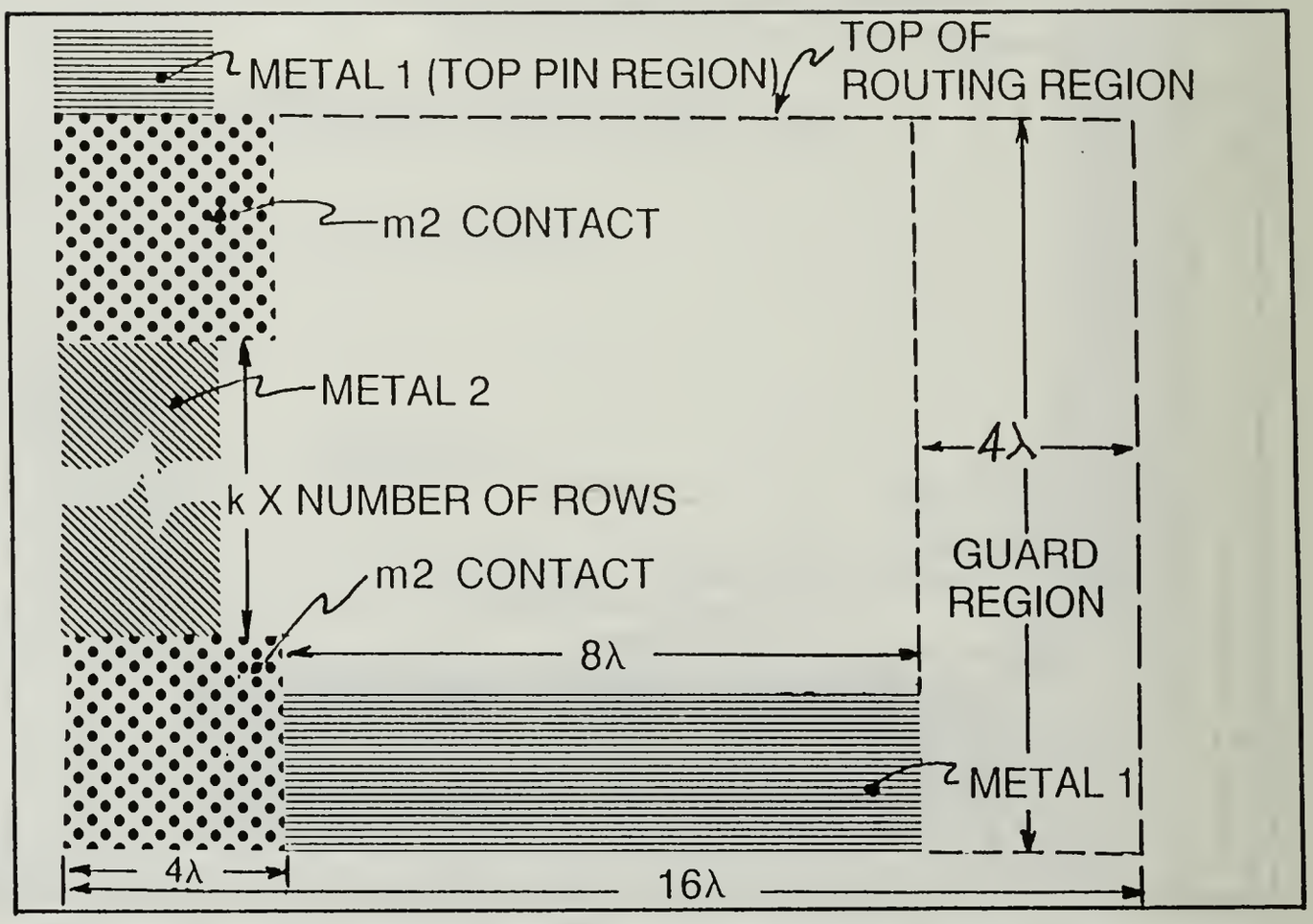

Figure 14. Case3 - Pin Entrance From Top of Channel: Entrance into channel with pin being metal1 and adjacent track net is not equal to pin number. In this case k is equal to K which is the value selected by the user (see section 3 page 25). The value k is then multiplied by the number of rows to produce the required length of metal2 vertical connector needed to reach the free track.

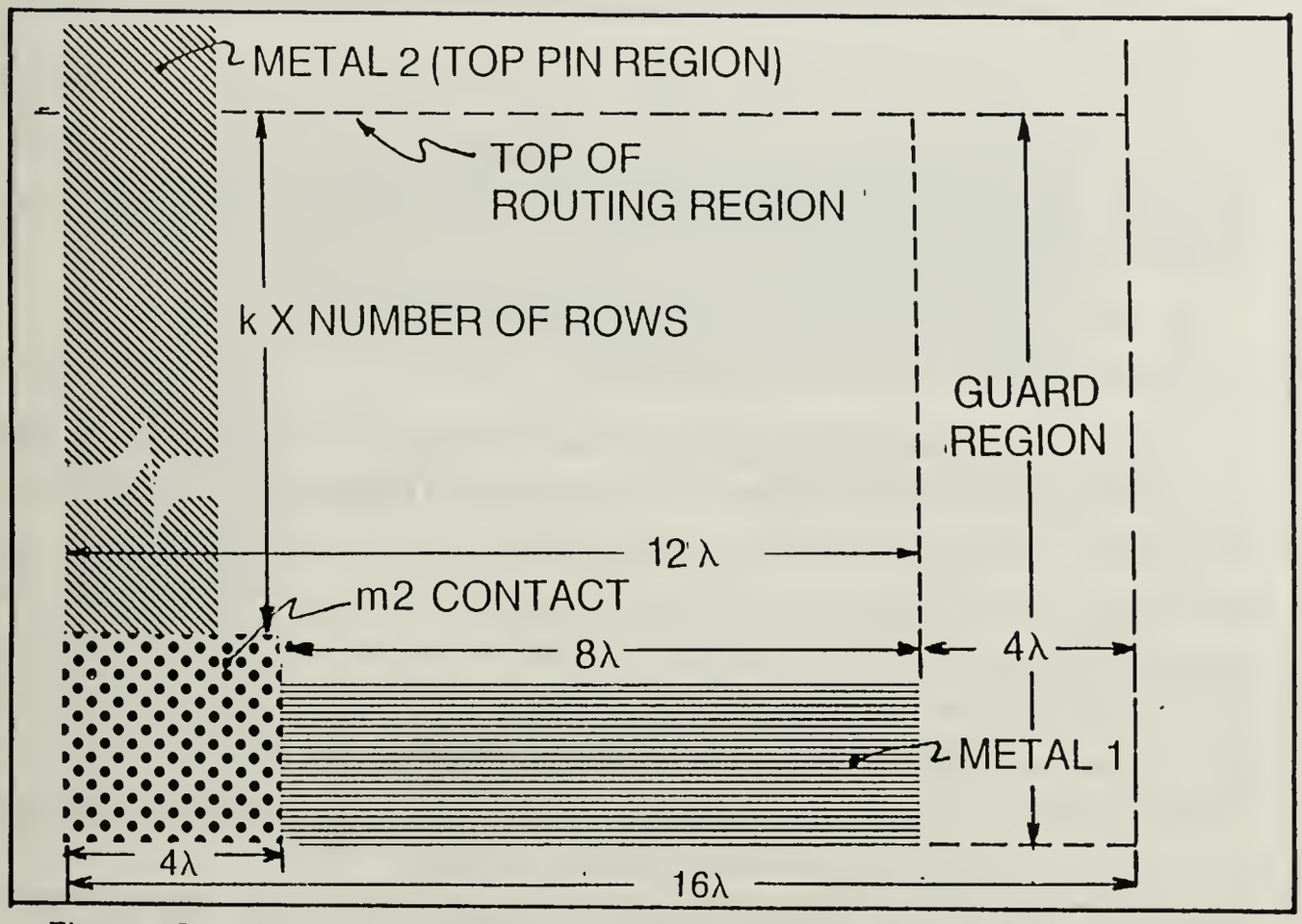

Figure 15. Case4 - Pin Entrance From Top of Channel: Entrance into channel with pin being metal2 and adjacent track net is not equal to pin number. In this case k is equal to K which is the value selected by the user (see section 3 page 25). The value k is then multiplied by the number of rows to produce the required length of metal2 vertical connector needed to reach the free track.

6. Left Scan Area Bottom Pin Routing

The next phase of the routing process is to complete the routing in the left scan area for the bottom pins. This is accomplished by first scanning from the column with the maximum density to the leftmost boundary of the routing region and asking the following question at each column:

- Is there a track already occupied having the same net number as the pin?

If the answer to this question is yes, the pin then establishes the connection to this track (see Figures 16 through 19), thereby joining the appropriate net.

After the left scan is complete, there may still remain some bottom pins which were not routed. The algorithm then scans from column one of the routing region to the column where the left scan was first initiated and ask two essential questions at each column in the following order:

1. Is there a free track?
2. Has the channel capacity been exceeded?

If the answer to the first question is yes, the algorithm allows the pin to make a connection at this track (see Figures 16 through 19). The track that is selected is the one closest to the bottom channel borders and also unoccupied. The algorithm then looks ahead in the left scan area and into the right scan (i.e., the area to the right of the column with maximum density) area and checks all the top and bottom pin net numbers with the new track net number. The algorithm then extends the track out to the right-most column having the same pin net number on either the top or the bottom of the routing region. At this point in the algorithm the net now occupies a track in the range [a, b], where a, b, have been previously defined (see page 27 left scan top pin routing). Unlike the top left scan routing, at this point there is no need to look back into the left scan area because the scanning has been proceeding from column one to the column of maximum density.

Finally, if the answer to question 1 was no then the algortihm sends a message indicating the channel is full and the density must be increased.

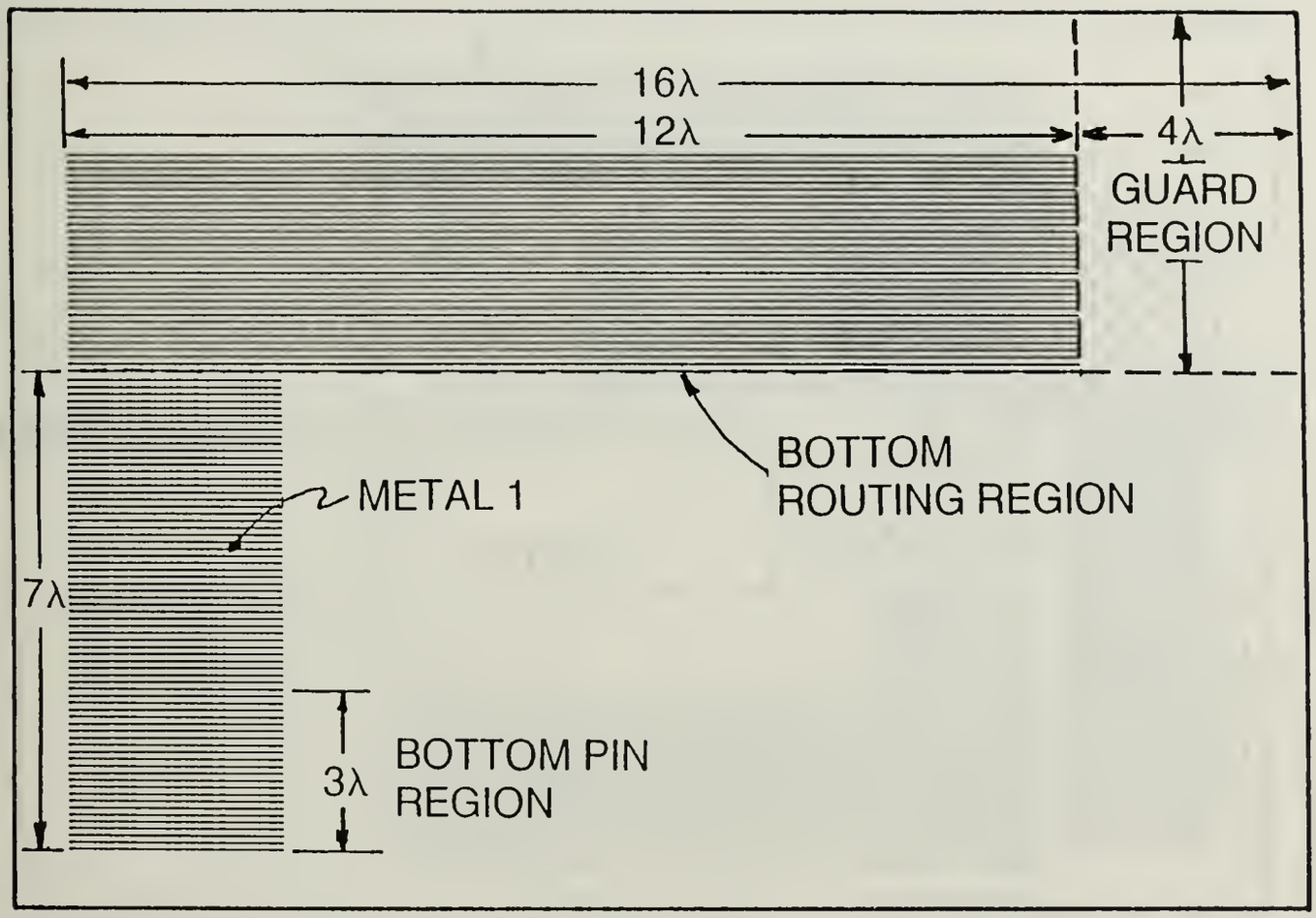

Figure 16. Case1 - Pin Entrance From Bottom of Channel: Entrance into channel with pin being metal1 and adjacent grid track is equal or unoccupied.

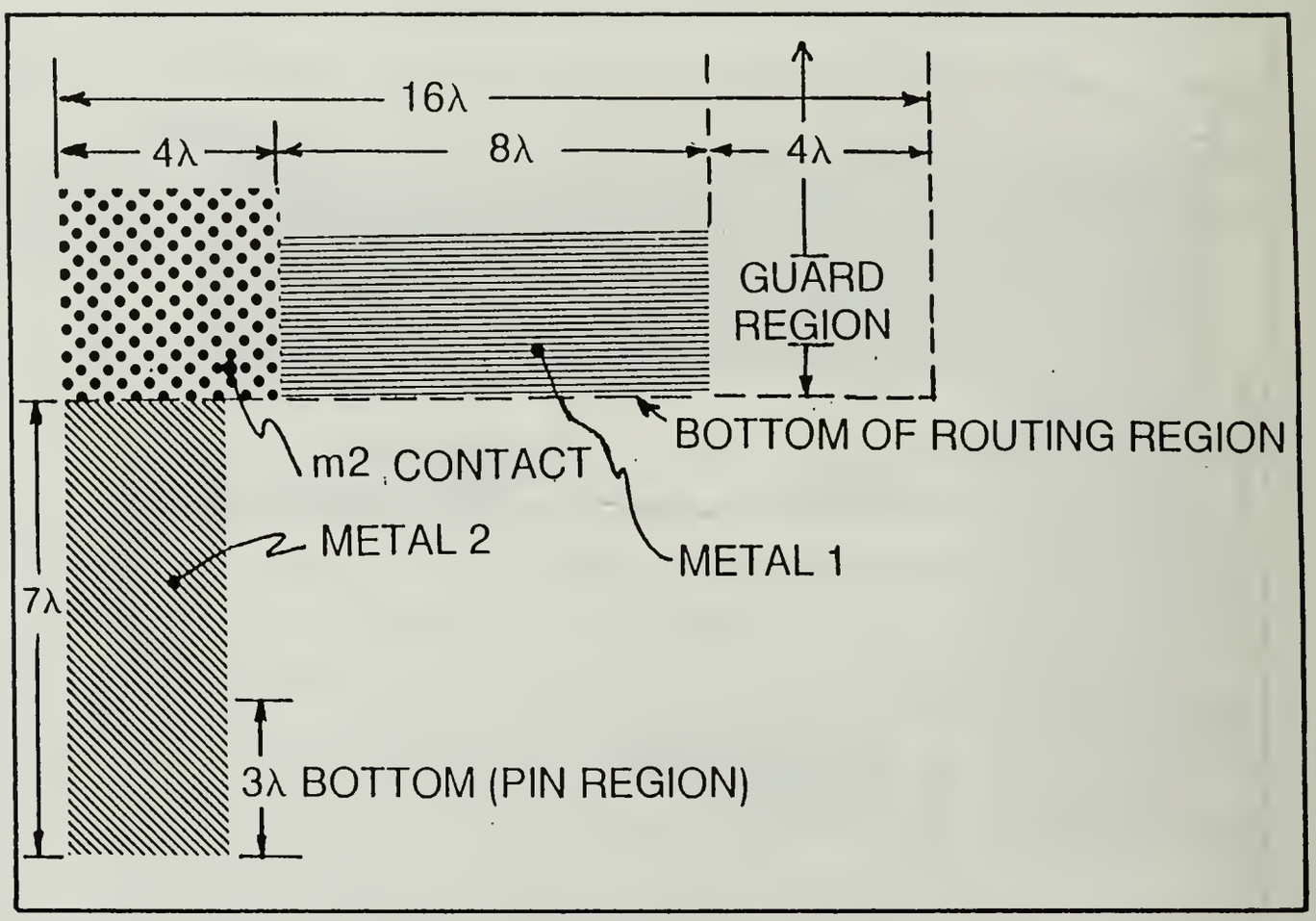

Figure 17. Case2 - Pin Entrance From Bottom of Channel: Entrance into channel with pin being metal2 and adjacent grid track is equal or unoccupied.

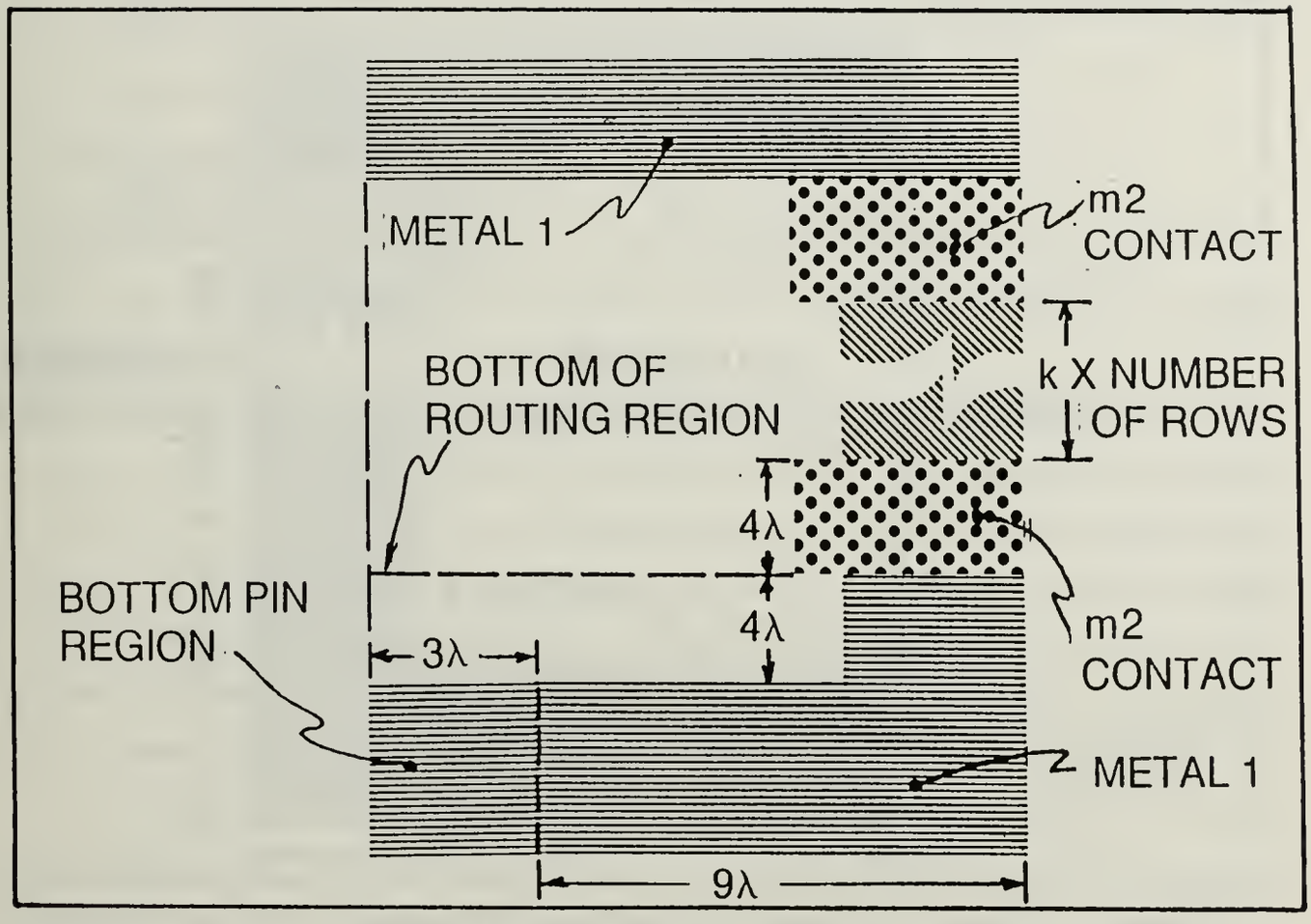

Figure 18. Case3 - Pin Entrance From Bottom of Channel: Entrance into channel with pin being metal1 and adjacent track net is not equal to pin number. In this case k is equal to K which is the value selected by the user (see section 3 page 25). The value k is then multiplied by the number of rows to produce the required length of metal2 vertical connector needed to reach the free track.

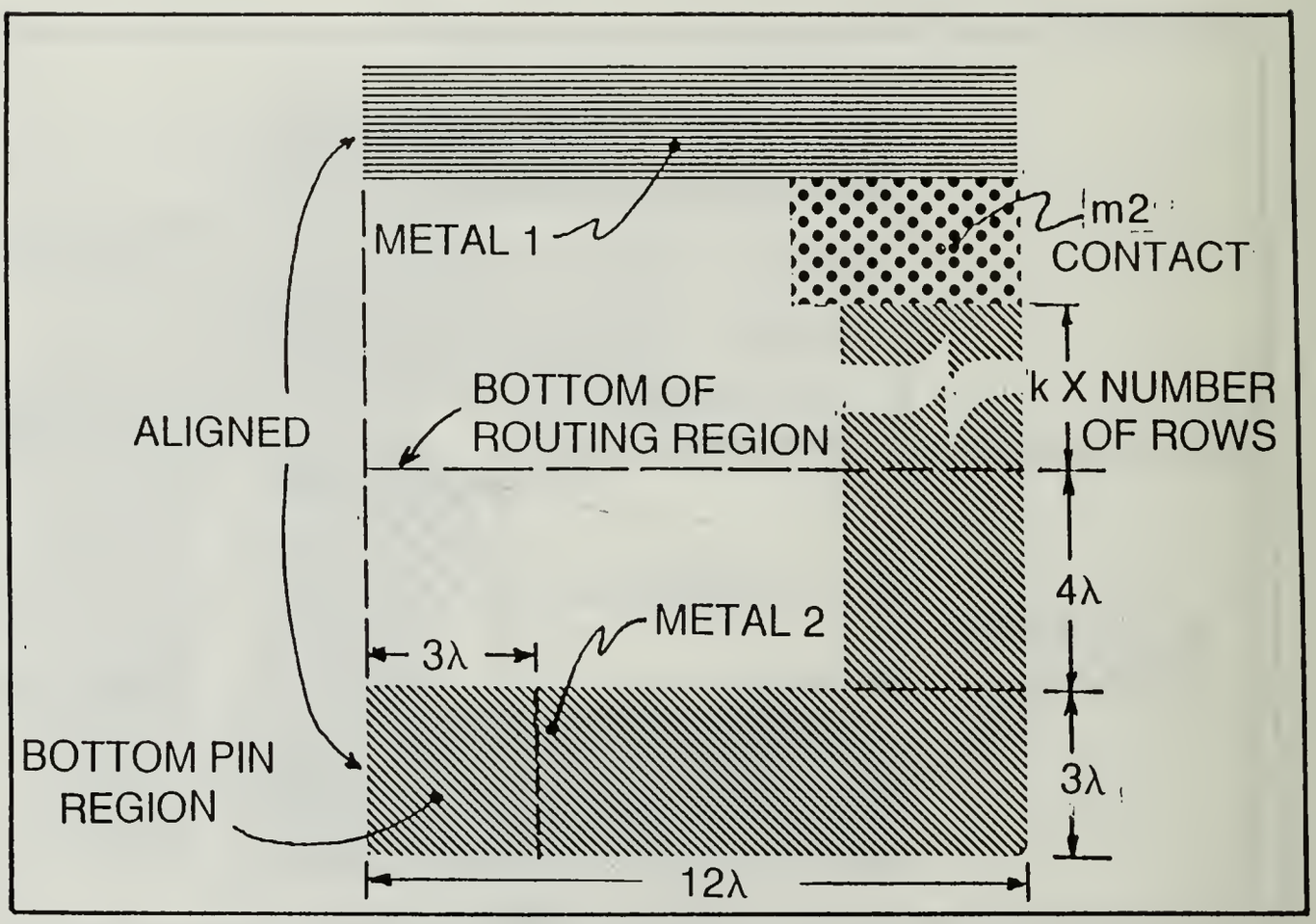

Figure 19. Case4 - Pin Entrance From Bottom of Channel: Entrance into channel with pin being metal2 and adjacent track net is not equal to pin number. In this case k is equal to K which is the value selected by the user (see section 3 page 25). The value k is then multiplied by the number of rows to produce the required length of metal2 vertical connector needed to reach the free track.

7. Right Scan Area Top Pin Routing

The next step in the routing process is to enable the top pins in the right scan area to make their initial entrance into the routing region. This phase is very similar to the routing of the pins in the top left scan area. However, there are some important differences. For this reason the following procedure is presented in detail. The router now asks three essential questions for each pin at every column in the following order:

1. Is there a track already occupied having the same net number as the pin attempting to make an initial entrance into the routing region?
2. Is there an empty track?
3. Is the routing region fully occupied?

If the answer to question one is yes then the pin moves to this track and connects. The reader is encouraged once again to refer to Figures 12 through 15, to better understand how this is actually accomplished.

If the answer to the first question is no and the second question is yes then the pin moves into the routing region and occupies a new track. The track which is occupied is the one that is closest to the top channel borders. As was the case in section 5 of this chapter, this is accomplished by using one of the schemes depicted in Figures 12 through 15. The scheme which is used is dependent upon the type of material the pin is and if the adjacent grid track is free or not free. Once the new track is established the algorithm looks "ahead" at all pins (top and bottom) in the right scan area only, and extends this new track to the net's rightmost boundary in the routing region. The rightmost boundary being a column which has either a top pin or bottom pin net number the same as the net number of the pin currently being routed. The router also looks "behind" in the right scan area only, and only at the bottom pins. The router then extends this new track to the net's leftmost boundary. The leftmost boundary in this case being a column which has a bottom pin net number the same as the net number of the pin that is currently being routed. At this point in the algorithm the net now occupies a track in the range [a, b], where a, b have been previously defined (see page 27 left scan area top pin routing).

If the answers to questions 1 and 2 were no, then the algorithm sends a message indicating the channel density has been exceeded and must be increased. The router is now ready to start the routing of the bottom pins in the right scan area.

8. Right Scan Area Bottom Pin Routing

The last phase of the channel routing is the routing of the bottom pins in the right scan area. This phase is accomplished in manner very similar to the way routing of the bottom pins in the left scan region was achieved. However, because of some important differences, the procedure is presented in detail. The routing at this phase is accomplished by first scanning from the column directly to the right of the column with the maximum density over to the rightmost boundary of the routing region and asking the following question at each column:

- Is there a track already occupied having the same net number as the pin?

If the answer to this question is yes, then the pin establishes the connection to this track (see Figures 16 through 19), thereby joining the appropriate net.

After this right scan is complete, there may still remain some bottom pins which were not routed. The algorithm then scans again from the column directly to the right of the column with maximum channel density over to the rightmost column in the routing region and ask two essential questions at each column in the following order:

1. Is there a free track?
2. Has the channel capacity been exceeded?

If the answer to the first question is yes, the algorithm allows the pin to make a connection at this track (see Figures 16 through 19). The track that is occupied is the one which is closest to the bottom channel borders. The algorithm then looks ahead in the right scan area and checks all the top and bottom pin net numbers with the new track net number. The algorithm then extends the track out to the rightmost column having the same pin net number on either the top or the bottom of the routing region. At this point in the algorithm the net now occupies a track in the range [a, b], where a, b have been previously defined (see page 27 left scan top pin routing).

Finally, if the answer to question 1 was no, then the algortihm sends a message indicating the channel is full and the density must be increased.

9. Programming And Source Code

All programming of the algorithm was done in the C language by the author. This language provided the necessary flexibility for providing easy implementation of large structures, such as the "MAP_REG" structure (see Appendix B and page 25 section 3). The entire program code for the router is listed in Appendix B. The main calling program is called *arouter.c*. The programs which update grid coordinates as the routing

progresses, begin appropriately with the word *update*. Programs beginning with the word *rowfind* are used to find an empty track. The program called *message.c*, sends a message to the user indicating the channel density has been exceeded when the channel is full. The file called *globe.h* is a file which contains "global" type variables that are used throughout the code. The file called *makefile* enables quick and easy recompilation after corrections to programs have been made, and greatly saves on compiler time. All programming was done on the VAX 11/785. The total code length is 4320 lines.

The program structure is such that, following initialization, arouter.c begins the left scan sequence as described in sections 5 and 6. Arouter.c determines which of the various update programs to call depending on how the questions in the algorithm sequence are answered. The update programs serve two basic functions. First, they update the grid coordinates as the routing progresses. These grid coordinates represent the actual X and Y cartesian coordinates of the nets, with respect to the grid structure. The coordinates of X = 0 and Y = 0 would specify the bottom left point of the grid. The nets and their respective tracks are made up of a series of rectangles each with assigned grid coordinates. The second function of the update programs is to perform the actual painting of these rectangles. The painting is done according to the layer type (see Figures 9 through 11). After the left scan sequence is completed, arouter.c commences the right scan sequence in accordance with sections 7 and 8. Again, arouter.c will decide on which of the various update programs to call depending on how the questions in the algorithm sequence are answered. The update programs in turn may call the rowfind programs when searching for a free (i.e., unoccupied) track.

10. Similarity And Differences To Greedy

Like Greedy the algorithm does assume that all pins and wiring lie on a common grid. Also similar to Greedy the algorithm assumes vertical wiring is done on one layer and horizontal on another. The algorithm is similar to Greedy in that for a net entering the channel for the first time, a new track is established whenever possible. The algorithm also uses the channel density value as an optimum starting place for deciding the depth of the channel, similar to the Greedy method. The algorithm does scan in a column by column method which is similar to the Greedy Algorithm.

The algorithm does not however route both bottom and top pins at every column like Greedy. The algorithm instead routes top and bottom pins separately as a "subset" in each scan area. The algorithm does not concern itself with split nets because a track is always made available or else the channel density has to be increased. The algorithm also does not concern itself with a steady net constant (see Chapter II page

5), because a multiple net uses the same track. Finally, the algorithm does not recognize rising or falling nets, but rather the algorithm routes to the nearest empty track.

In this thesis, the primary concern was to obtain some results as to how varying the start scanning position would affect the routing. With this in mind, some of the parameters used in the original Greedy algorithm though still considered viable were relaxed, while others were restricted. This was done to expedite the research and ease some of the already heavy programming burden.

11. Input/Output

The initial routing area with pins, net numbers and unrouted channel area is stored in a file of type magic called *temp.mag*. The results of the router are stored in a file called *route.mag* and the program code to generate this magic file is listed in the appendix under the code titled *out.c*. The program is user interactive and allows for changing such parameters as pin layers, channel density and the start scanning position. The user is also able to create a new data file (i.e., new routing problems). There are two ways to create a new data file. The first is through a user interactive program called *input.c*. The main calling program will ask the user if a new data file is desired. If the user wants to create a new data file then the program interaction begins and the data file is stored in a file called *temp.data*. The user may then restart the router and use this new data file as input. The second method is by using a program called *initialize.c*. This program is independent of the main router code and has the same structure as input.c. The data file is also stored in a file called temp.data. Appendix A contains a detailed procedure on how to use the router.

12. Switchbox Routing And Algorithm Extension

Following the successful routing of the channel problem, the next goal of this thesis was to formulate a continuation of this new algorithm to enhance the capabilities of the router for the switchbox problem. We now describe the steps which will enable the routing of a switchbox when incorporated into the existing channel algorithm. This extension will make use of the new channel routing algorithm as much as possible.

Just like the top and bottom pins, the router will evenly space the left pins and right pins along the left and right sides of the switchbox. Therefore, for every left pin there must be a corresponding right pin. If a pin is not to be routed then it is given the net number of zero. Left and right pins are evenly placed along the left and right edges of the routing region with a 4λ (due to SCMOS rules and the DRC) guard region placed in between the pins and the first and last columns of the routing region. The pins are placed and aligned with the bottom line of their respective grids.

The following is a step-by-step program outline of the switchbox extension algorithm:

1. **SW1:** Read in necessary inputs.
2. **SW2:** Calculate new channel density for the switchbox - D_{sw}.
3. **SW3:** Establish tracks for nets with left and right pins only. Repeat this step for all such tracks. This step is not required if a net has either top pins, bottom pins or both.
4. **SW4:** Route according to channel sequence (see sections 5 through 8) and target row technique (see step four below). Scanning is done according to the channel routing algorithm using the four area scanning method.
5. **SW5, SW6, SW7:** Connect all remaining left and right pins.

The SW is an abbreviation for the word switchbox.

The algorithm extension assumes it always has enough area to do the routing. Consequently if the area is not available the routing will not be 100%. This assumption is not unusual; routers such as Detour also work under this premise.

The first step (**SW1**) in this modification would require changes to the input phase of the routing process. This would be accomplished by the modification of the program input.c (see Appendix B) to allow for the data entry of the left(i) and right(i) pins, where "i" is the pin number of the left and right pins. Such data would include the pin's net number and layer type. The program arouter.c would also have to be altered to allow for the user to specify whether the problem is a channel or switchbox. If the problem is specified as being a switchbox then the value of K (see section 3 page 25) would have to be set at a minimum of 19λ.

The second step (**SW2**) would be to enable the algorithm to compute a new channel density value based on the left(i) and right(i) pin net numbers. This new channel density is designated D_r . The channel density is defined to be the maximum number of nets which have pins on both sides of the line $x = \alpha$ for any α. To find D_r, the channel would now be rotated 90° (see Figure 1). In this case the pins that are used in the computation are the left(i) and right(i) pins from column 1 to column N, where N is the rightmost column in the routing region. The number of columns are now determined by the number of left or right pin numbers. The program number.c can be utilized to compute this new channel density. Now the algorithm has to compare this channel density value, D_r, to the density value computed for the top(i) and bottom(i) pins, referred to as D_c. The algorithm would then choose the greater of the two values. Therefore, the new channel density $D_{sw} = \max(D_r, D_c)$. D_{sw} is the value to which the rows should

be initialized. With this final channel density value computed, the pins can now be placed along the left and right edges of the routing area in preparation for routing.

The third step (**SW3**) would examine all nets around the switchbox. If a net has pins located on the left and right sides only, then the routing sequence would proceed in the following manner:

1. Establish a track along the bottom edge of the row at which the net's topmost pin is located and connect the respective pin to this track. The length of this track is determined by the criteria delineated in steps 2 and 3. If there is a conflict, then the router places a track at the location of the left(i) pin. A conflict would arise if a left(i) and right(i) pin both occupied a topmost position. The net's topmost pin in this case is the pin located closest to the top of the routing region. The pin would have to be extended by 4λ because of the existing guard region.
2. If the net has pins on the left side only, the length of the track would be limited to one grid (i.e., 12λ). Note the actual grid width is 16λ, but this also includes the 4λ guard region. This track would be located in the grid adjacent to the pin. Similarly, the same procedure would apply if a net has pins on the right side only.
3. If the net has pins located on both the left and right sides of the routing region, then the length of this track would span the entire routing region and would be equal to the number of columns multiplied by 16λ.

This step (**SW3**) is repeated until all nets in the above mentioned category are routed.

The fourth step (**SW4**) in the algorithm uses a scheme similar in nature to Luk's switchbox [Ref. 7: pp. 6-7], where the following additional constraints are imposed on the algorithm:

1. Connecting the pins on the left of the routing region.
2. Connecting the pins on the right of the routing region.

The similarity to Luk lies in the use of what are called target rows. During the channel routing portion of the routing, the algorithm was not concerned with what specific tracks the pins were initially allowed to occupy. However, for the switchbox routing this is not the case. When routing the switchbox, the algorithm uses the same sequence as that of the channel, but must ask two additional questions at each of the four scan area phases, due to the additional constraints (left and right side). They are as follows:

1. Is there a left(i) pin net number the same as the top or bottom pin net number which is currently making its initial entrance into the routing region?
2. Is there a right(i) pin net number the same as the top or bottom pin net number which is currently making its initial entrance into the routing region?

- If the answer to questions one and two is yes, then the router will allow the pin (top or bottom) to jog to the track corresponding to the location of the left(i) pin. The left(i) pin can now be extended out to this track. Assuming the right(i) pin lies on the same row as the left(i) pin, then the length of the track would be equal to the

number of columns multiplied by 16λ. However, if the right(i) pin does not lie on the same row then the track is extended out to the column containing the net's rightmost boundary (see section 6).

- If the answer to question one was yes and question two was no, then the router will allow the pin (top or bottom) to jog to the track corresponding to the location of the left(i) pin. The left(i) pin can now be extended out to this track. The track is now extended out to the column containing the net's rightmost boundary (see section 6).
- If the answer to question one was no and question two was yes, then the router will allow the pin (top or bottom) to jog to the track corresponding to the location of the right(i) pin. The right(i) pin can now be extended out to this track. The track is now extended out to the column containing the net's leftmost boundary (see section 5).
- If the answer to both questions were no, then the router will continue according to sections 5 through 8, depending on which scan area the routing is currently taking place in.

The fifth step (**SW5**) in the algorithm extension would be to connect all pins on the right side of the routing region which require no vertical connecting jogs. These are the right pins which have a track located at a distance of 4λ to their left. The pins merely have to extend their tracks by 4λ.

The sixth step (**SW6**) would connect remaining pins whose corresponding net had pins located only on the right side, left side or both. This step would be accomplished by routing all left side pins from top to bottom and then, if necessary, the right side pins from top to bottom. The pins would have to make an upward vertical jog (to avoid conflicting with existing tracks), center on the adjacent grid and finally enter the routing region at the adjacent column. Now the column (first or last) would have to be checked to insure there is enough vertical jogging space for each pin with a different net number. If there is not, then the column width would have to be increased appropriately to insure that space for a 3λ vertical connector is allowed and also a 4λ spacing distance on both sides of this connector is allowed. One should recall that the column width initially available for jogging is a maximum of 12λ (see section 3). However, this will be further reduced if there are top or bottom pins which were previously routed along the left or right sides of the grid. Once the jogging space is available, then the pins can jog to their respective tracks.

The seventh (**SW7**) and final step in the algorithm extension would connect all remaining pins. Specifically these are left(i) or right(i) pins belonging to a net which has at least one top pin, bottom pin or both. As was the case in step 6, the sequencing will be from the top to bottom and left side to right side. The pins would again jog upward

vertically (to avoid existing track conflict), center on the adjacent grid, enter the routing region, and jog to the nearest column belonging to their respective net number. If a cyclic conflict (between same track left(i) and right(i)) should arise, then the pin which will make the longer jog would have priority to use the track for jogging. The other pin would then be required to jog down or up (depending on the location of the pin's net track) to the next free track space and continue jogging until a column connection can be made. A cyclic conflict would also necessitate increasing the current column (first or last) width by the method delineated in step 6.

IV. TESTING AND RESULTS

A. GENERAL

1. Channel Routing

The new router has been named the NPGS router. All results for the NPGS router are presented in this section. Comparisons of various routing problems were done in two different ways. First NPGS was compared against itself for various "start" scanning columns. Secondly comparisons were made against MAGIC's router Detour. All comparisons made against MAGIC were done using the identical routing area as that used by the NPGS router. Unfortunately, detailed comparisons could not be made against the original Greedy algorithm because the program code for the algorithm was not available. All experiments were conducted on the VAX 11/785 and using an AED 787 colorware monitor graphics device. The wire length is based on units of 1.5 microns/λ.

2. Switchbox Routing

This phase of the testing was accomplished by handrouting the test examples using the algorithm extension set forth in section 12 of Chapter III. The reason for this was due to the time constraints involved to do the actual programming. It was decided that only two small routing test would be conducted. Here, again the tests were conducted by comparing NPGS against MAGIC and by comparing NPGS against itself. All comparisons made against MAGIC were done using the identical routing area as that used by the NPGS router and using the same equipment as that used for the channel routing test.

B. CHANNEL ROUTING COMPARISONS WITH MAGIC

As stated under the objectives of this thesis, one of the goals was to be able to route Deutsch's channel. The outward scanning router was capable of routing Deutsch's channel [Ref. 4: p. 423] and Burstein's difficult channel [Ref. 16: p. 637]. Another goal was to achieve 100% routing. The NPGS router achieved 100% routing of both the Deutsch's and Bursteins's difficult channel. With regards to Deutsch's channel the NPGS router did beat MAGIC's router Detour in the via category by almost a factor of two. The reason for this might be due to the obstacle avoidance capability that Detour posseses. Another reason may be due to the fact that Detour is capable of handling cyclical conflicts (see page 7). This feature sometimes necessitates more layer changes

and therefore more vias. The NPGS channel router does not have to negotiate cyclical conflicts, because nets with multiple pins occupy the same track. In the wire length category Detour won out by a considerable margin. For the Deutsch's channel test the NPGS router channel density setting was set at 19 and the NPGS router started scanning in column 13. A complete and detailed listing of the routing results for Deutsch's channel can be seen in Table 2. The actual routing by both NPGS and MAGIC of Deutsch's channel can be seen in Figures 20 and 21.

Table 2. NPGS ROUTER VS THE DETOUR (MAGIC'S) ROUTER EXPERIMENT1

CHAN-NEL PROB-LEM	ROUTER	NO.of ROWS	NO.of COLS	NO.of VIAS	WIRE LENGTH
Deutsch's	NPGS	19	38	64	11208
Deutsch's	DETOUR	19	38	124	10501

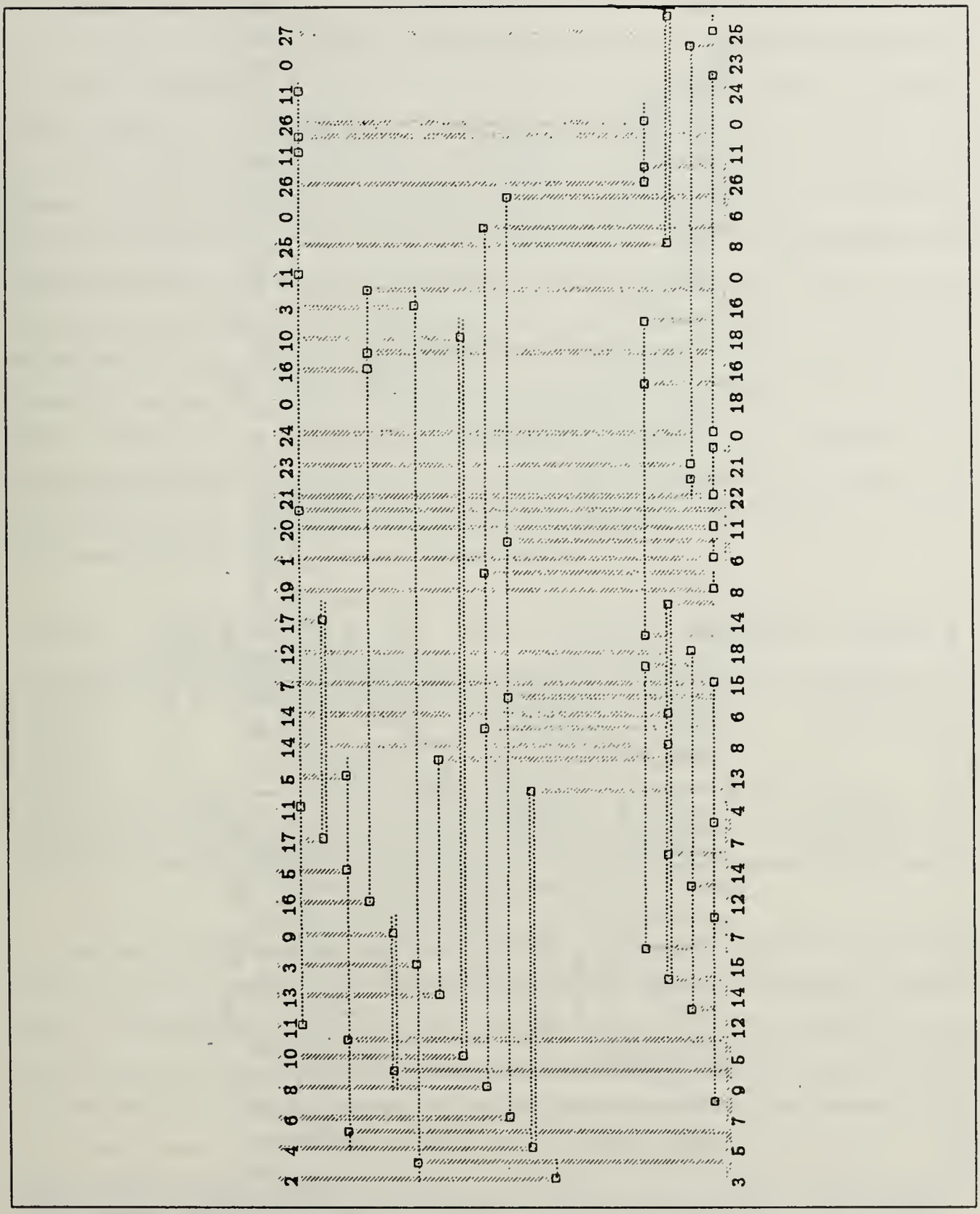

Figure 20. Experiment1 - NPGS solution to Deutsch's difficult channel: Greedy based algorithm with outward scanning technique. Channel density at 19 and start scan setting at 13.

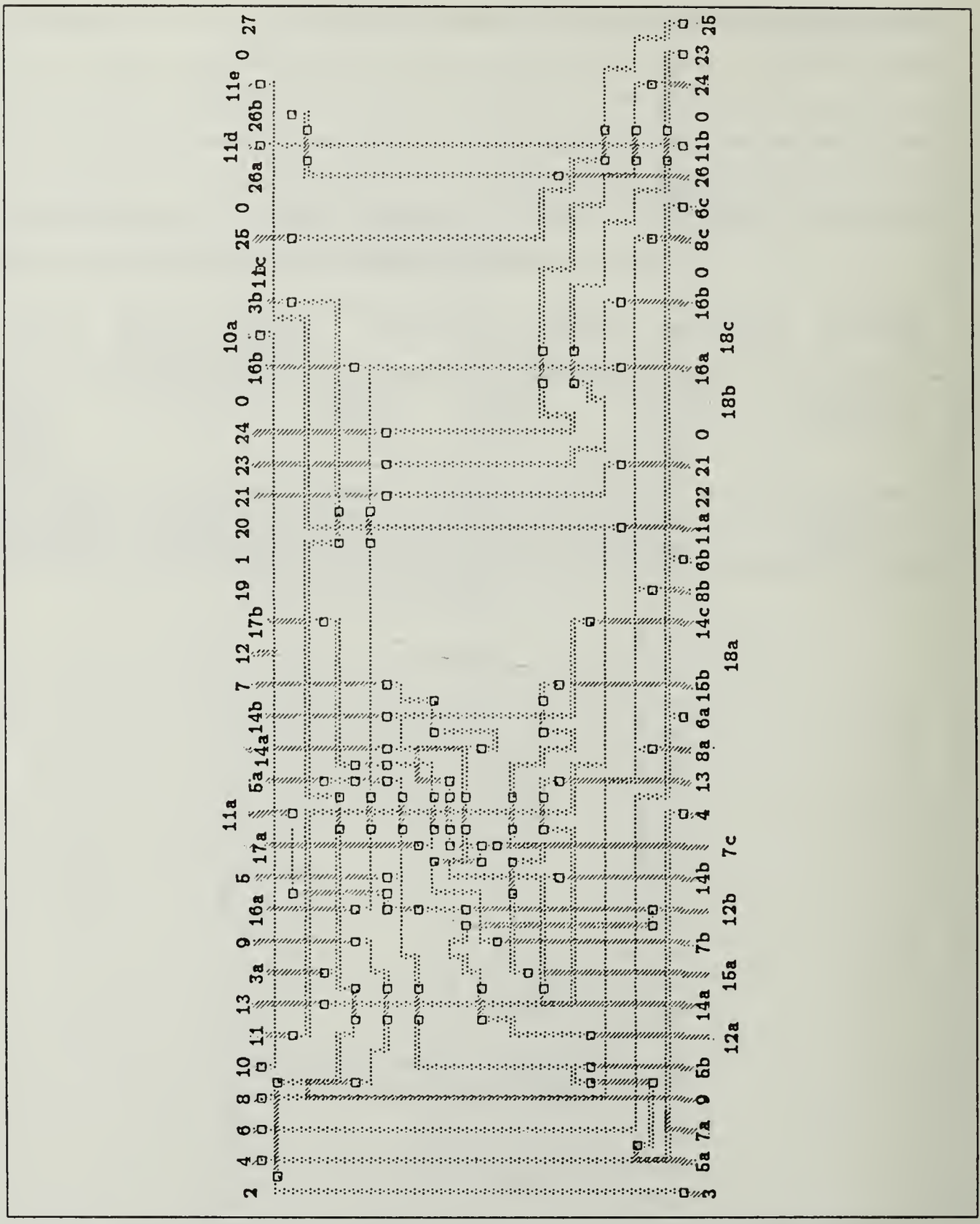

Figure 21. Experiment1 - Detour's solution to Deutsch's difficult channel: The channel has the same area as that of the NPGS routing solution. The routing was done net by net from left to right.

The reader should note that for routing purposes MAGIC does not allow for the use of the same pin numbers in a cell. Therefore, some nets are numbered with subscripts a, b, c, etc. (see Figures 21 and 23). The next test to be conducted was that of Burstein's difficult channel. In this test the NPGS router used slightly more vias than that of the MAGIC and in the wire length category MAGIC again beat NPGS but only by a small amount. A possible reason for the closeness in the number of vias may be contributed to the fact that the jog length used by Detour was set close to the channel width. MAGIC however does produce considerably more bends than does NPGS. As previously mentioned, the increase in bends may contribute to slower circuitry. It should also be pointed out that Burstein's channel has only a third the number of pins which the Deutsch's channel has. With reference to the Burstein's channel, the NPGS channel density setting was set at 5 and the NPGS router started scanning in column 3. A complete and detailed listing of the routing results for Burstein's channel by both NPGS and MAGIC can be seen in Table 3. The actual routing by both NPGS and MAGIC of Burstein's channel can be seen in Figures 22 and 23.

Table 3. NPGS ROUTER VS THE DETOUR (MAGIC'S) ROUTER EXPERIMENT2

CHANNEL PROBLEM	ROUTER	NO.of ROWS	NO.of COLS	NO.of VIAS	WIRE LENGTH
Burstein's	NPGS	5	13	25	1357
Burstein's	DETOUR	5	13	22	1310

One reason for the increase in wire length by NPGS could be due to the fact that MAGIC does not restrict itself to wiring horizontally in metal1 and vertically in metal2, where as NPGS does. By allowing either metal1 or metal2 in any direction, the same column can sometimes be used for multiple nets, by overlapping layers, thereby reducing the wire length and still meeting MOSIS rules. Another reason for the increase in wire length may be due to the fact that NPGS does not use a rising and falling net scheme like that of the Greedy router.

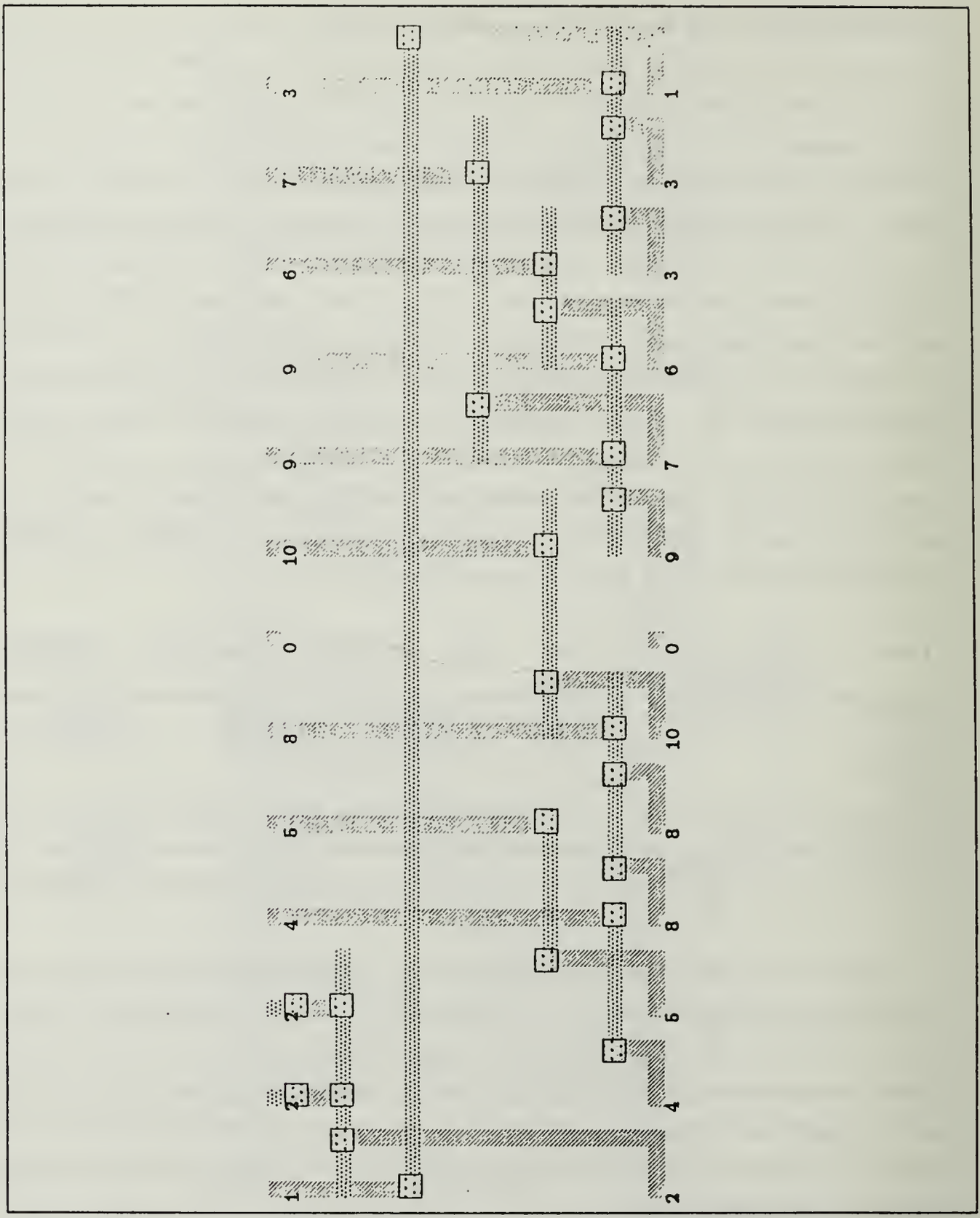

Figure 22. **Experiment2 - NPGS solution to Burstein's difficult channel:** Greedy based algorithm with outward scanning technique. Channel density at 5 and start scan setting at 3.

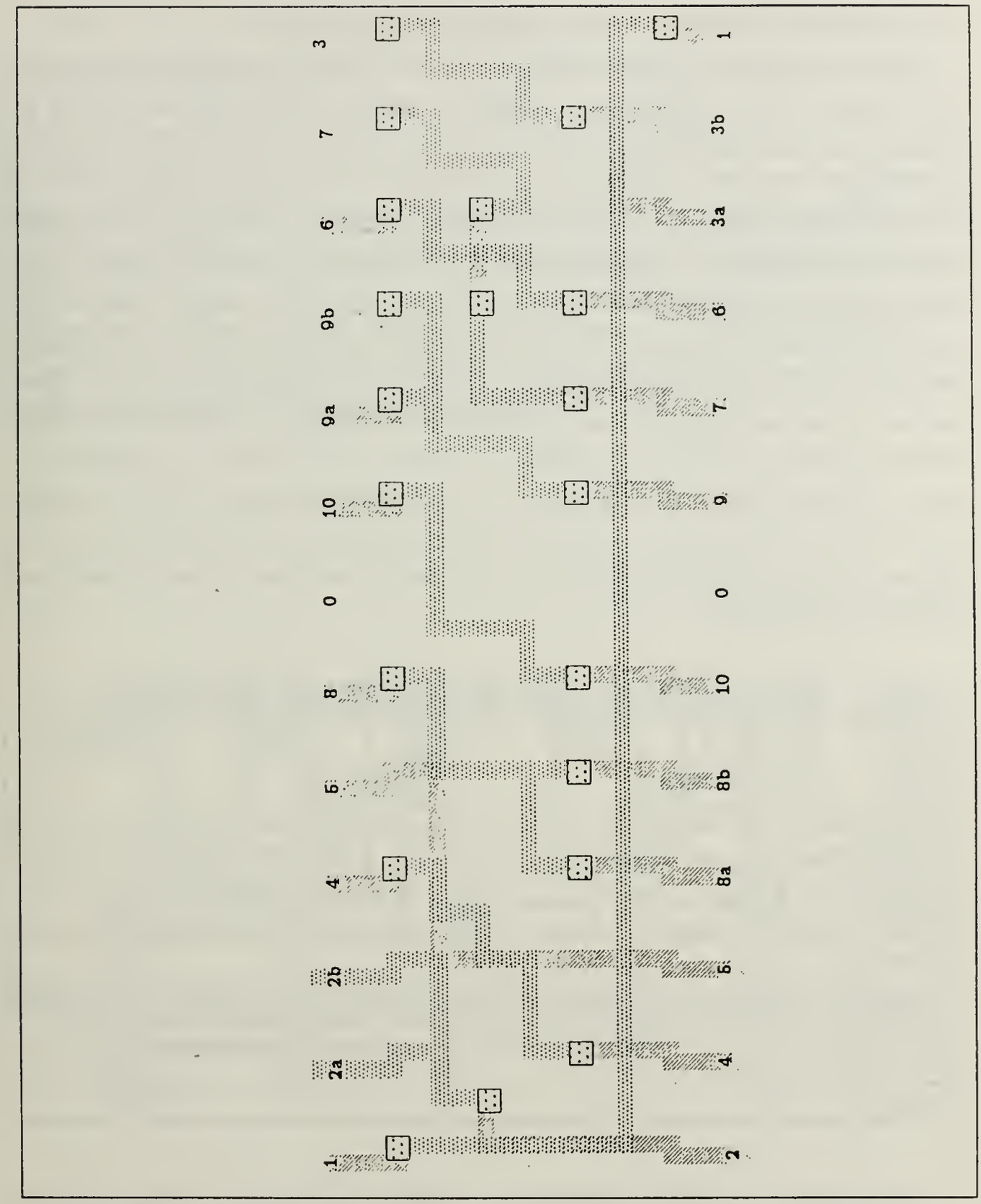

Figure 23. **Experiment2 - Detour's solution to Burstein's difficult channel:** The channel has the same area as that of the NPGS routing solution. The routing was done net by net from left to right.

C. CHANNEL ROUTING COMPARISONS WITH NPGS

The final phase of channel routing tests to be conducted consisted of comparing NPGS against itself. This involved varying the channel density and also varying the column from which the scanning actually begins. The tests were conducted on Burstein's and Deutsch's channel. The start scanning column was allowed to alternate from a column of maximum density to column number one for two different test runs on each of the two test channels. The channel density was kept at the algorithm value for one run of each of the channels and was then reduced to the lowest routable value. We did this for both the Burstein's and Deutsch's channels. The algorithm value of the channel density is the value which the program code actually computes based on the given number of pins and corresponding net numbers (see Appendix A number.c). The reader should note also that these are the lowest possible channel density settings for both of the test channels which the algorithm could use and still achieve 100% routability. Notice that we were able to reduce the Burstein's channel density from 5 to 4 and the Deutsch's channel density from 19 to 14. The results are shown below in Table 4 and Figures 24 through 29.

Table 4. NPGS ROUTER VS NPGS ROUTER CHANNEL RESULTS

CHAN-NEL PROB-LEM	START COL# DENSITY#	NO.of ROWS	NO.of COLS	NO.of VIAS	WIRE LENGTH
Burstein's	3 5	5	13	25	1357
Burstein's	1 5	5	13	25	1357
Burstein's	3 4	4	13	25	973
Burstein's	1 4	4	13	25	973
Deutsch's	13 19	19	38	64	11208
Deutsch's	1 19	19	38	69	12318
Deutsch's	11 14	14	38	69	7708
Deutsch's	1 14	14	38	69	8818

The results for the first and fifth entries in Table 4 are shown in Figures 20 and 22. The NPGS router was unable to completely route Deutsch's channel with a density setting of 14 while still starting at a column of maximum track density. With a channel density setting of 14 and a start column of 13 one net could not be completely routed. Therefore, the Deutsch's test for a channel density of 14 used a start column of 11, which

is a column with a channel density of 12 (see the seventh entry from the top of Table 4). The results for Burstein's problem shows the NPGS outward scanning method with the same results as the NPGS starting in column one (i.e., scanning left to right). The Deutsch's channel test showed the NPGS router with outward scanning winning in the wire length category for both tests. The reason for this might be because the outward scanning method starts with a column of maximum density and therefore allows the nets with several pins to make an earlier channel entrance and therefore occupy a track closer to the channel borders. The shorter the track distance is from the pin, the shorter the overall routing wire length will be. The number of vias that were generated did not show any significant differences. Possible methods for improving the router's performance in the wire length category will be discussed in Chapter V.

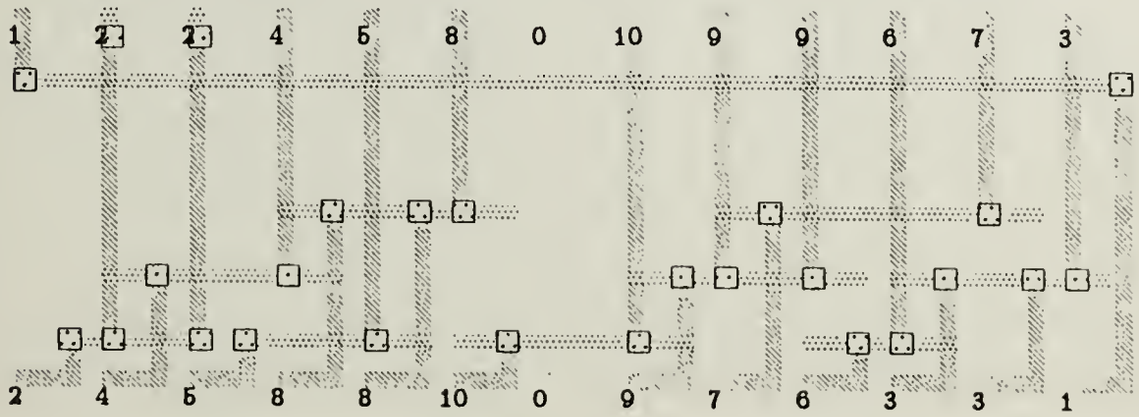

Figure 24. Experiment3 - NPGS solution to Burstein's difficult channel: Greedy based algorithm with outward scanning technique. Channel density at 5 and start scan setting at 1.

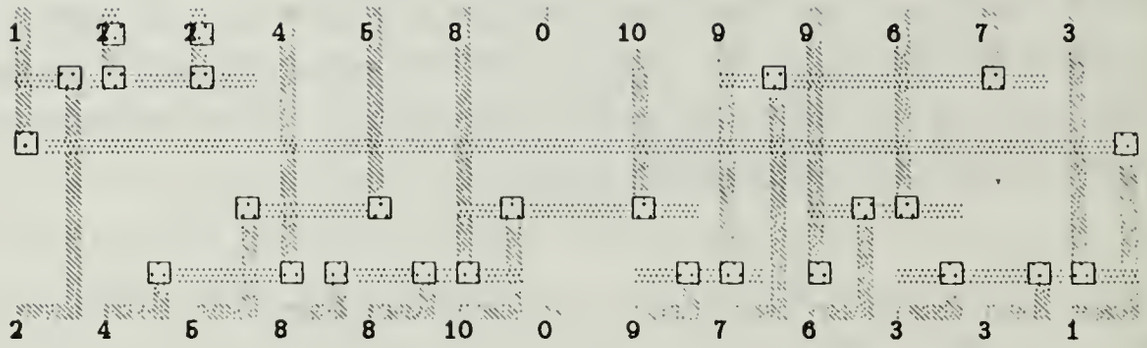

Figure 25. **Experiment4 - NPGS solution to Burstein's difficult channel:** Greedy based algorithm with outward scanning technique. Channel density at 4 and start scan setting at 3.

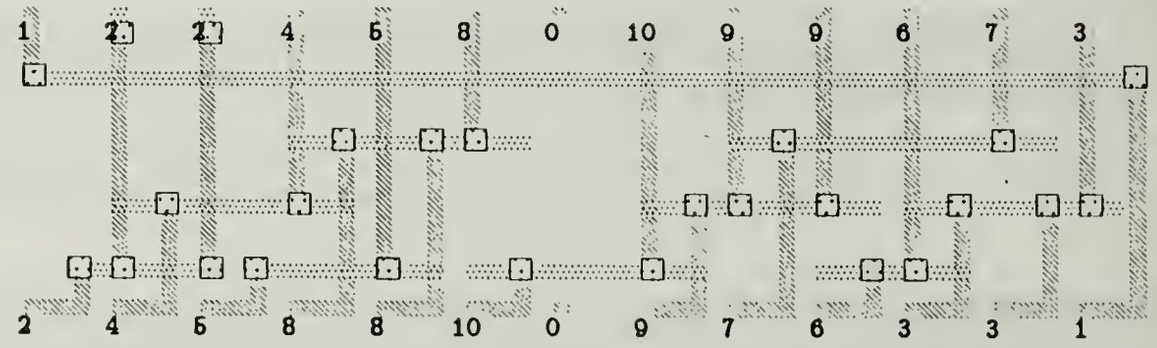

Figure 26. **Experiment5 - NPGS solution to Burstein's difficult channel:** Greedy based algorithm with outward scanning technique. Channel density at 4 and start scan setting at 1.

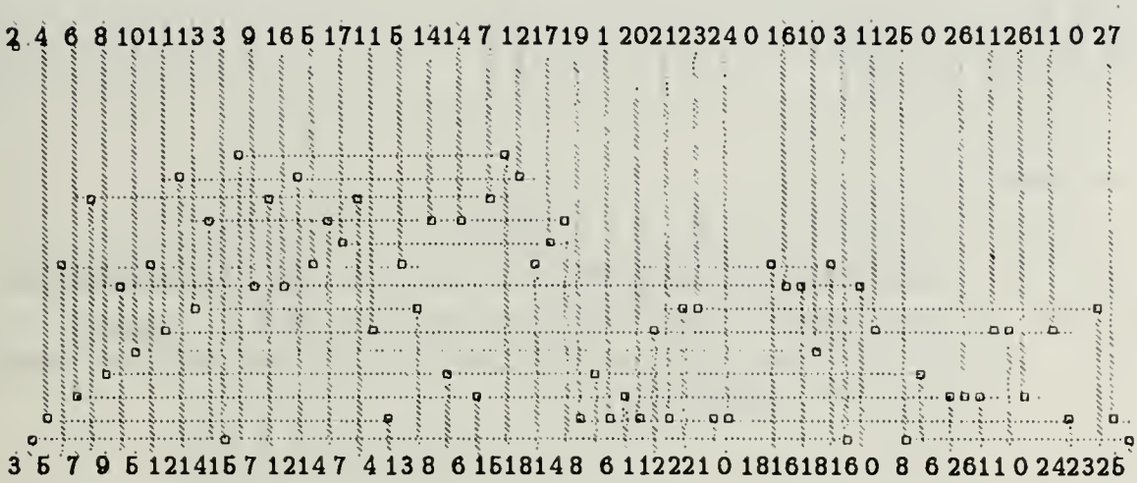

Figure 27. Experiment6 - NPGS solution to Deutsch's difficult channel: Greedy based algorithm with outward scanning technique. Channel density at 19 and start scan setting at 1.

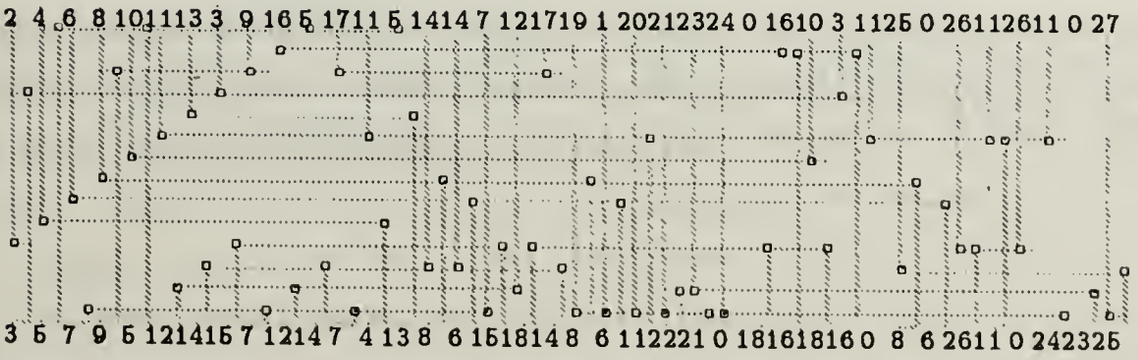

Figure 28. Experiment7 - NPGS solution to Deutsch's difficult channel: Greedy based algorithm with outward scanning technique. Channel density at 14 and start scan setting at 11.

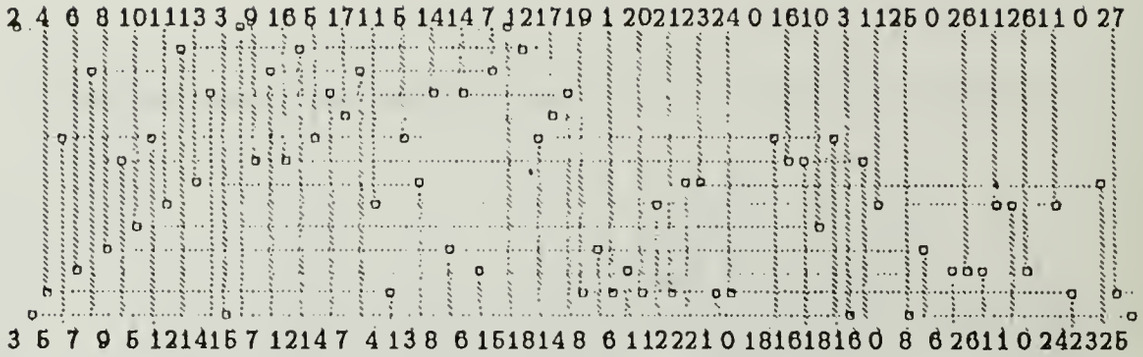

Figure 29. **Experiment8 - NPGS solution to Deutsch's difficult channel:** Greedy based algorithm with outward scanning technique. Channel density at 14 and start scan setting at 1.

D. SWITCHBOX ROUTING COMPARISON WITH MAGIC

This phase of the testing was designed to prove that the algorithm extension (see section 12 of Chapter III) does indeed work and can be implemented as an automatic switchbox router. The switchbox test that was used was a 48 pin/12 net problem. The problem contains 12 pins along each of the routing area borders and each of the 48 pins belongs to one of 12 nets. The routing problem is considered "moderate" in terms of difficulty. This type of test was selected because the NPGS router had to be "hand routed". The hand routing was done by following the switchbox algorithm extension. Handrouting is accomplished by using a "mouse" device in conjunction with the CAD tool and individually wiring each net. The NPGS router was able to find a 100% solution and used considerably less vias than MAGIC. MAGIC's wire length was less than that of NPGS. MAGIC, however, did not find a 100% solution to the problem. The reader should note that MAGIC was unable to completely route the pins along the top edge of the routing area corresponding to nets 9, 10, 11 and 12. The router (Detour) was allowed to route the switchbox starting from left to right and then from right to left. In both attempts, MAGIC was unable to complete the routing. The nets that MAGIC did successfully route, were done net by net and automatically. A complete listing of the routing results for the 48 pin/12 net test is shown in Table 5. The actual routing results are shown in Figures 30 and 31.

Table 5. NPGS ROUTER VS THE DETOUR (MAGIC'S) ROUTER EXPERIMENT9

SWITCH-BOX PROB-LEM	ROUTER	NO.of ROWS	NO.of COLS	NO.of VIAS	WIRE LENGTH	% Rout -abil -ity
48pin/12net	NPGS	12	12	40	6045	100%
48pin/12net	DETOUR	12	12	67	5904	83%

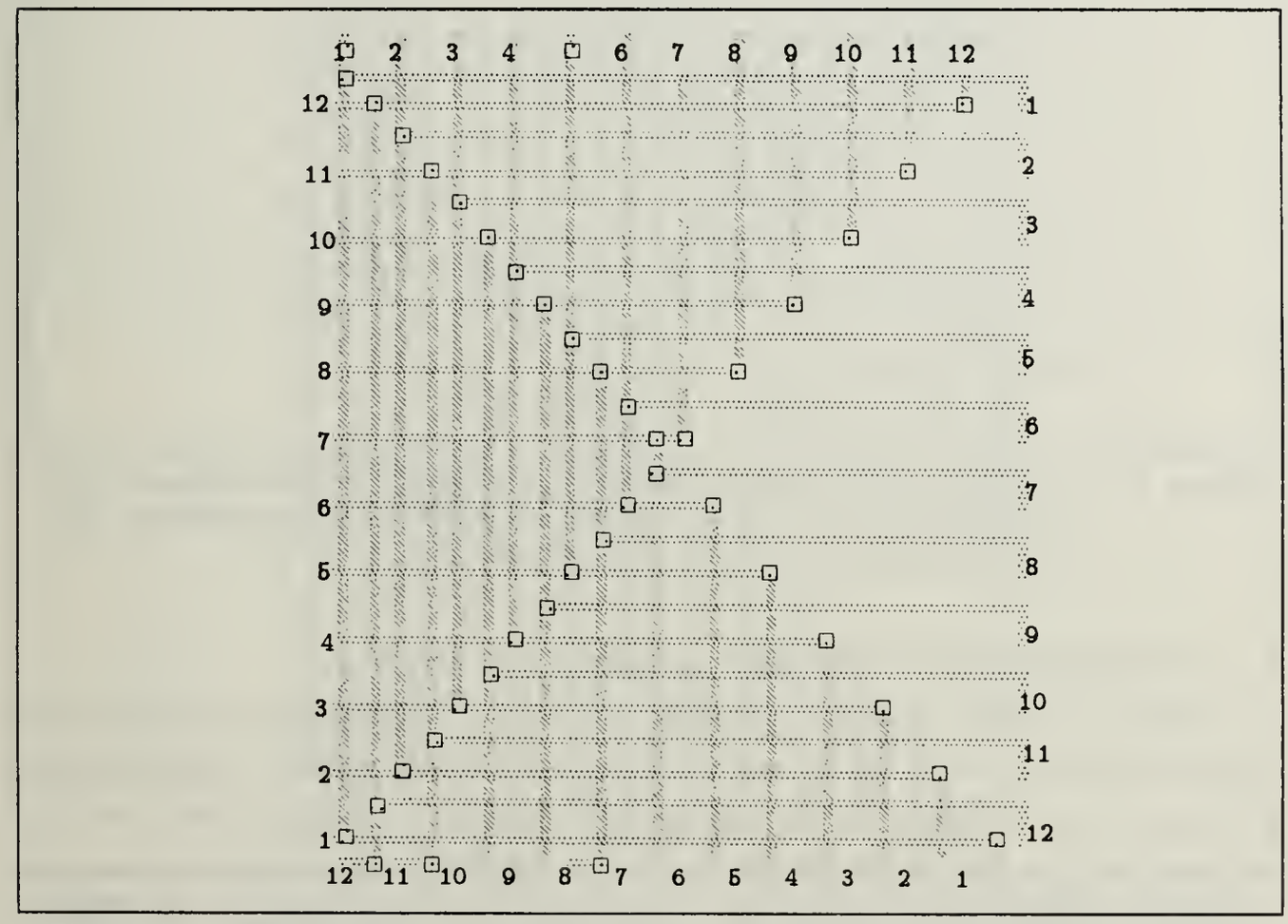

Figure 30. Experiment9 - NPGS solution to the 48pin/12net switchbox: Greedy based algorithm with outward scanning technique. Density equal to 12 and start scan setting at 7.

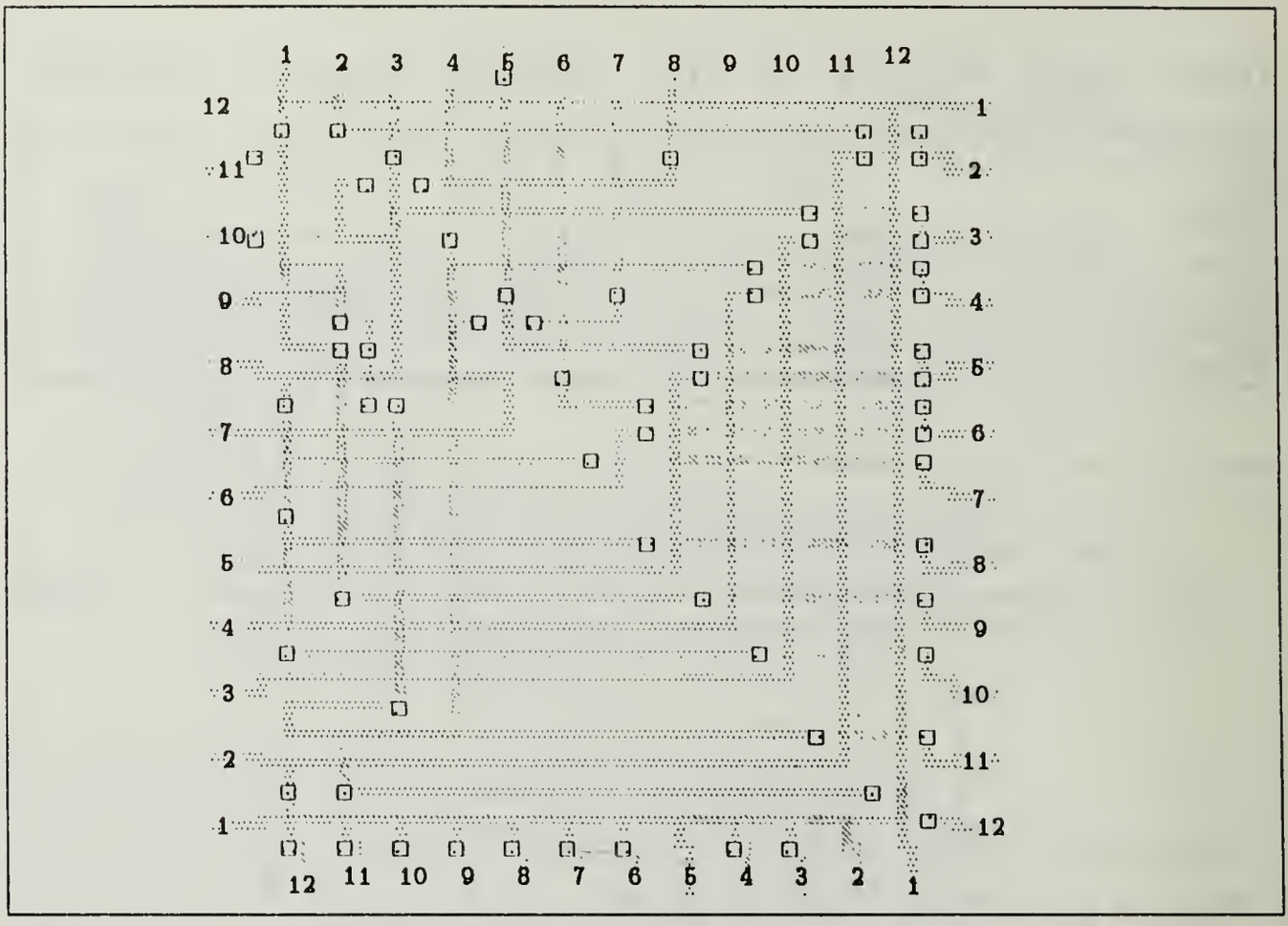

Figure 31. Experiment9 - Detour's solution to the 48pin/12net switchbox: The switchbox has the same area as that of the NPGS routing solution. The routing was done net by net from left to right.

E. SWITCHBOX ROUTING COMPARISON WITH NPGS

This test involved comparing NPGS against itself. The comparison was made between the results of the 48 pin 12 net switchbox test that was routed by NPGS (shown in Figure 30) and routing the same problem, this time with a different starting position. For this test the scan was initiated in column one. The results showed the via count and wire length were identical for both test. This is due to the fact that the NPGS router employs a target row concept (see section 2 Chapter III). By using this target row technique, all the top pins were "targeted" for the same rows for both tests. The results are listed in Table 6 and shown in Figures 32 and 30.

Table 6. NPGS ROUTER VS NPGS ROUTER SWITCHBOX RESULTS

SWITCH-BOX PROB-LEM	START COL#/ DENSITY#	NO.of ROWS	NO.of COLS	NO.of VIAS	WIRE LENGTH
48pin/12net	7/12	12	12	40	6045
48pin/12net	1/12	12	12	40	6045

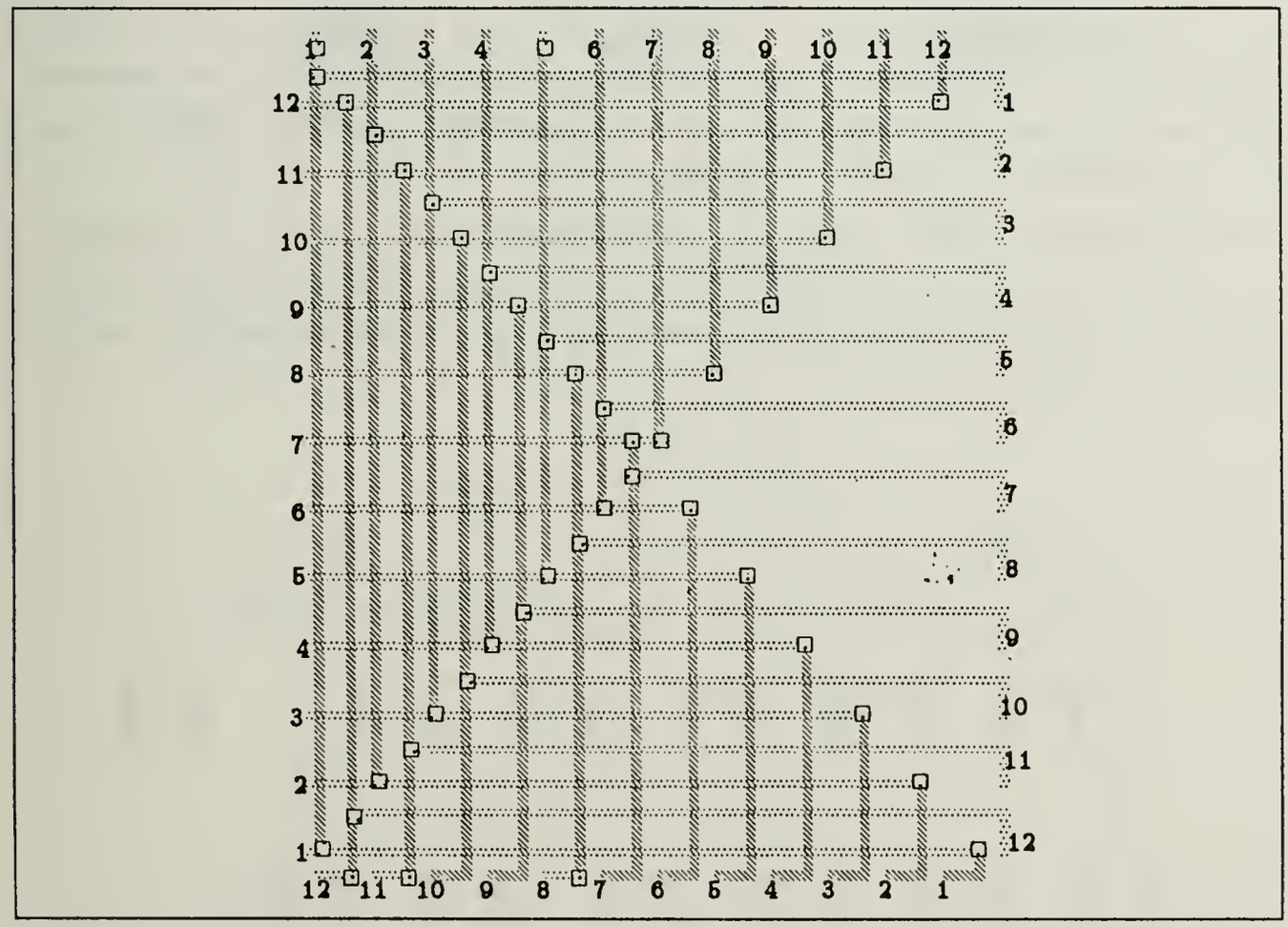

Figure 32. Experiment10 - NPGS solution to the 48pin/12net switchbox: Greedy based algorithm with outward scanning technique. Density equal to 12 and start scan setting at 1.

F. THE ROUTER AS A CAD TOOL

This section was inserted to demonstrate how the router can be used as a CAD tool. Figure 33 depicts a typical VLSI layout design. The design contains 3 nand gates along the top of the figure, a channel in the middle, and 12 inverters at the bottom of the figure. The 3 nand gates may typically be contained in one cell and referred to as cell 1. Similarly the inverters are referred to as cell 2. The area in the center is the channel. The reader can see that the channel has 24 pins which are subsequently combined into 12 nets. Once the routing problem is specified, the user can make a third cell (automatically) by utilizing the NPGS router. This third cell would contain the solution to the routing problem. The ordering of the pins along the channel is left up to the discretion of the user. In order to complete the design process the user only has to generate the stems from cell two and cell one to cell three.

This example may seem trivial, but if the number of pins and nets were doubled in size, one can see how a hand routing solution would be very laborious and tedious. The NPGS router offers an easy method of producing "cell" solutions to complex routing problems.

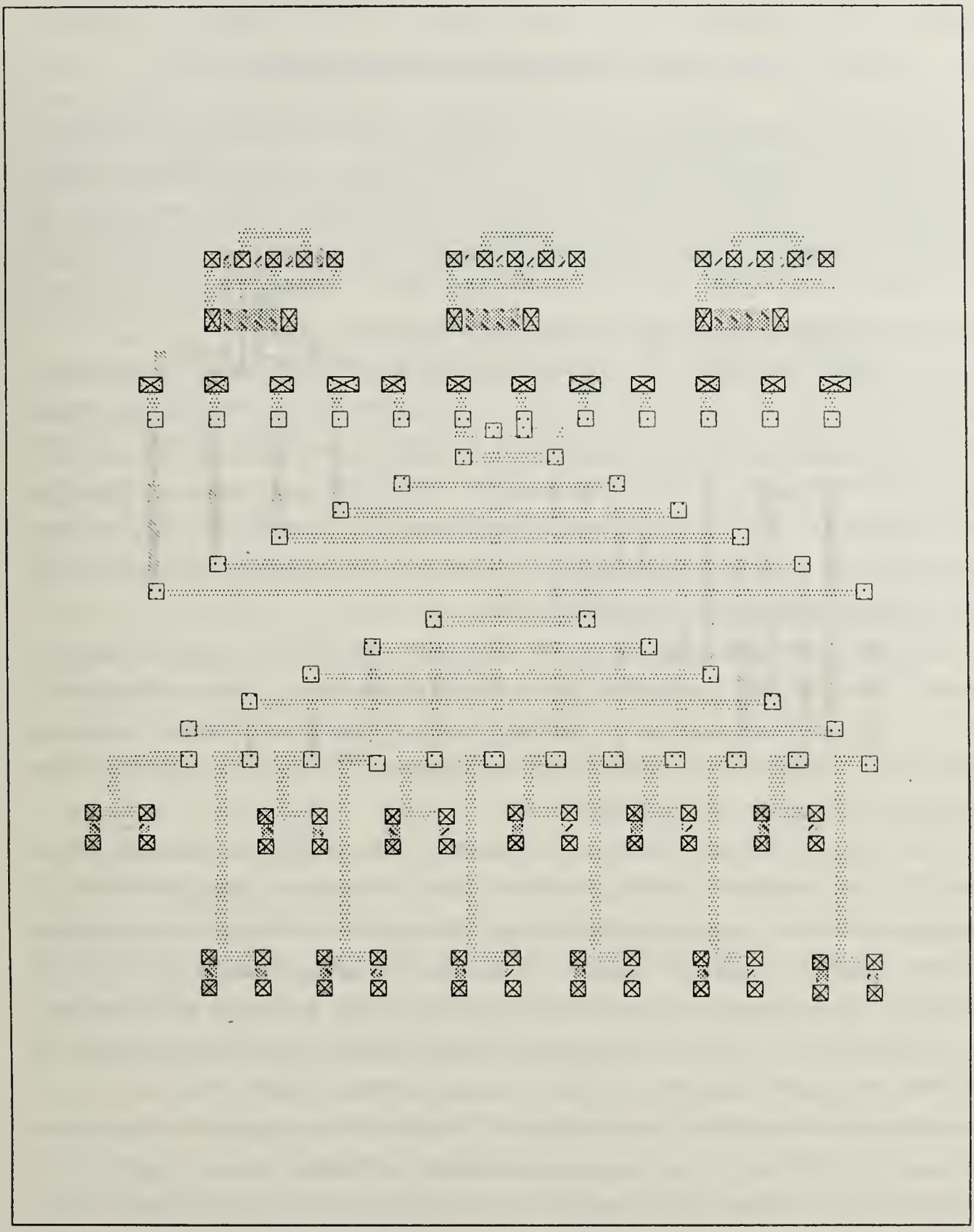

Figure 33. NPGS Used As A VLSI CAD Tool: Channel routing of a VLSI design done by the NPGS router.

V. CONCLUSION AND DISCUSSION

The outward scanning method offers yet another feasible approach to VLSI routing. The tests in this thesis have shown that the NPGS router, whether scanning outward or scanning in a traditional method can be used to route difficult routing problems. The outward scanning technique does in fact produce overall better results, especially in the wire length category. The router offers the flexibility of starting in any column, using any channel density setting, and choosing the best results.

The NPGS router was able to reduce or maintain the number of vias generated compared to that of MAGIC. In the Deutsch's channel test the via reduction was as high as 50%. The wire length increased from 3%-10% above that of MAGIC. MAGIC only allows routing one net at a time manually. For the channel routing problem, this router allows all the nets to be routed automatically. This means that after the input parameters and routing problem have been specified, the NPGS router automatically creates the routing scheme for the given area.

Another interesting conclusion is the advantages and disadvantages of grid based routers versus gridless routers. Grid-based routers are easier to program conceptually, but do not offer much flexibility for changes. Gridless routers offer a more intelligent approach to routing, but are more difficult to program. Finally, the router can also be used as a CAD tool in the VLSI course.

An important parameter which can be inserted and probably have the most noticeable affect on the channel routing results would be to consider a rising and falling net as explained below. As shown by the testing in Chapter IV, by taking into consideration a rising/falling net, we can decrease the channel density. This could conceivably be accomplished by inserting a new step in the algorithm, where a new pin would occupy a track depending upon where the next pin in the net were located. If the next pin in the net was a top pin (a rising net), then the entering pin would occupy a free track as close as possible to the top of the channel borders. Similarly, if the next pin in the net was a bottom pin (a falling net), then the pin would occupy a track as close as possible to the bottom channel borders. This rising net scheme might also be calculated on the basis of choosing the larger of rising or falling pins within a certain net. If a certain net had a predominance of top pins over bottom pins then the routing for that net would be done on a track as close to the top channel borders as possible. Similarly, the same

would hold true for a net with more bottom pins than top pins. Another method of reducing the wire length would be to use a third layer (e.g., polysilicon). This could be done by allowing either the top or bottom pins to be routed using polysilicon. This would eliminate the need for a 4λ guard region in each grid. The overlapping of polysilicon with metal2 is within MOSIS rules and would thus eliminate the number of vertical connectors. An evaluation of the routing geometry generated by NPGS when solving the Deutsch's channel problem, indicates that a 10% reduction in wire length is possible. However, adding a third layer type along with a second type of via will increase programming difficulties.

Other work on this thesis could include the development of a compaction routine, referred to in [Ref. 4: p. 422]. This routine would allow the router to "intelligently" reduce the track-to-track spacing and reduce the wire length if a pair of adjacent tracks does not have contacts next to each other in some column. This idea could also be used for adjusting column-to-column spacing. Future work on this thesis could also include the actual programming of the switchbox algorithm portion of this thesis. Additional work on the switchbox algorithm could include the capability to target both the left and right pins of the switchbox. In some cases this would entail a net initially occupying two separate tracks in the switchbox, where now it only occupies one. Future work on this thesis could also include the programming of a stem generator [Ref. 7: p. 148]. A stem generator would route the pins from the actual cells to the perimeter of the routing area. Work could also be done in actual cell routing using the NPGS router. Cell routing would allow for the use and experimentation of the routing in an actual VLSI chip or circuit design environment. Other parameters which were not implemented in this algorithm such as jog-length or the steady net constant could be incorporated in the algorithm to examine the affects they have on the routing. Also a program to automatically update the grid structure when the channel density is exceeded could be included in the algorithm. With this automatic update the program would not have to be restarted every time the density had to be changed.

Channel and switchbox routers are useful tools in the VLSI design process. There still remains countless new variations of heuristics to try. An idea which has been suggested in [Ref. 10: pp. 144-145] , concerns conducting research into the integration of an AI system such as Weaver and a greedy algorithmic system such as Magic. This type of integrated system would allow for choosing between time execution or overall routing performance, based on the individual's needs and desires.

APPENDIX A. NPGS ROUTER USER GUIDE

This appendix is designed to provide a detailed guide to using the NPGS router. Prior to actually using the router the user should know the basic pin configuration along each side of the channel routing area.The user should also know the net configuration eminating from the top and bottom channel cells. The ordering of the nets is not important and therefore flexibility is given to the user in this area. Figures 34 and 35 depict typical user sessions. Figure 34 shows a session after invoking the command *initialize*. Similarly, Figure 35 is typical after issuing the *route* command.

The first step is the creation of a data file. To create a data file the user types the command *initialize*. The user will then be prompted for a set of input data. The prompts for this data occur in the following order and are as follows:

1. ***Specify the number of terminals***. This is the number of pins along the top channel borders.

2. ***Enter the netlist number for the top(i) pin***. This is the net number for the corresponding top pin number i.

3. ***Enter the netlist number for the bottom(i) pin***. This is the net number for the corresponding bottom pin number i.

   Prompts 2 and 3 are repeated until all top and bottom pins and their corresponding net numbers have been entered in sequence.

   Now the router will compute the required channel density and initialize the MAP_REGION. The next series of prompts occur as follows:

4. ***Enter the netlayer for top(i)***.

5. ***Enter the netlayer for bottom(i)***. The netlayer refers to the type of material the pin is. The pin is either metal1 or metal2. The prompts 4 and 5 will repeat until all pins and their corresponding layers have been entered.

   Once all the data has been recorded. it is then stored in a file called temp.data and the user can rename this to any other convenient name. The initial channel area with pins not routed is contained in a file called temp.mag. The user is now ready to invoke the router. To run the router, the user types the command *route*. The router will prompt the user for the name of the data file by using the following statement:

6. ***Enter the name of the data file for the router***. The user then types in the appropriate name. The user will then see the router list the number of nets and display the channel density setting for the current channel. The user will then be asked the following question:

7. ***Would you like to change the channel density?*** If a new value is desired then the user now enters the number. The router will now display a complete lisitng of the pins and respective nets in the channel. The user is also shown the current start scan column and asked the next question:

8. *Would you like to change the start scan column?* The user can select a column to start the scanning from. The actual routing is now accomplished. Once this step is complete, the router will then ask one final question as shown below:
9. *Would you like to develop a new routing data file?* If desired, the user can now input a complete new routing problem.

Upon completion of the routing, the results with pins and routed channel is stored in a file called route.mag. The user should insure that the stems from both the top and bottom cells are evenly spaced by a distance of 12λ. The user can now establish a "parent" cell. The three cells (top, bottom, and routed) can be "dumped" into the parent cell to obtain a completed design configuration.

```
 % initialize


THIS PROGRAM ALLOWS THE USER TO CREATE A DATA FILE
FOR THE NPGS ROUTER
IT WILL BE STORED IN A FILE CALLED TEMP.DATA


specify the number of terminals n =  2

THE NUMBER OF TERMINALS IS  2

ENTER THE NETLIST # FOR TOP  1=1

ENTER THE NETLIST # FOR BOTTOM  1=2

ENTER THE NETLIST # FOR TOP  2=3

ENTER THE NETLIST # FOR BOTTOM  2=1
TRACKS PER COLUMN IS AS FOLLOWS

tracks = 0
tracks = 1
tracks = 0

THE CHANNEL DENSITY IS THE MAXIMUM
HORIZONTAL TRACKS =   1

enter the netlayer for top1:metal1
TYPE IN top1:top1

enter the netlayer for top2:metal2
TYPE IN top2:top2

enter the netlayer for bottom1:metal1
TYPE IN bot1:bot1

enter the netlayer for bottom2:metal1
TYPE IN bot2:bot2
```

Figure 34. Typical User Session Following The Initialize Command: Enables user to establish a data file for use with the router.

```
% route
ENTER NAME OF DATA FILE FOR CHANNEL ROUTER:  b.data
THE NUMBER OF NETS FOR THIS PROBLEM IS
n=  13
THE CHANNEL DENSITY SETTING IS CURRENTLY AT
MAX = 6

WOULD YOU LIKE TO CHANGE CHANNEL DENSITY?:       no

THE PIN NETS LOOK LIKE SO
top0= 0      bottom0= 0
top1= 1      bottom1= 2
top2= 2      bottom2= 4
top3= 2      bottom3= 5
top4= 4      bottom4= 8
top5= 5      bottom5= 8
top6= 8      bottom6= 10
top7= 0      bottom7= 0
top8= 10      bottom8= 9
top9= 9      bottom9= 7
top10= 9      bottom10= 6
top11= 6      bottom11= 3
top12= 7      bottom12= 3
top13= 3      bottom13= 1
top14= 0      bottom14= 0

STARTING SCAN IN COLUMN# = 3
WOULD YOU LIKE TO CHANGE THE START VALUE?:       no
WOULD YOU LIKE TO DEVELOP A NEW ROUTING DATA FILE?:       no
THE ROUTING IS OVER ___ BYE!!!!!!!!
```

Figure 35. Typical User Session Following The Route Command: Enables user to enter a data file and invoke the router.

APPENDIX B. C PROGRAM CODES

AROUTER.C

```
#include <stdio.h>
#include "globe.h"
#include <ctype.h>


/*****************************************************************************/
/*                                                                           */
/*    THIS IS THE BEGINNING OF THE                                           */
/*    CHANNEL ROUTER                                                         */
/*    INPUT IS DONE BY                                                       */
/*    A DATA FILE CONTAINING THE INPUT CHANNEL                               */
/*    THIS PROGRAM ALSO ALLOWS THE USER TO CREATE                            */
/*    A NEW DATA FILE.                                                       */
/*                                                                           */
/*****************************************************************************/




main()
{
int i,status,j,n,k,z,l,test;
int limit,rbound,count,keep,check,tcol;
int restart,netnum,dimension,check2;
FILE  *fp;






printf("ENTER NAME OF DATA FILE FOR THE ROUTER:  ");
gets(fname);



fp = fopen(fname,"r");

i = 0;
while(status = nextline(fp)){
       if (status == 1)
              i = i + 1,
             strcpy(list[i].ltr,buf);

          else
             i = i + 1;
```

```
            save[i] = num;

        }


metal1 = "metal1";
metal2 = "metal2";



  fclose(fp);
n = save[1];
printf("THE NUMBER OF NETS FOR THIS PROBLEM IS");
printf("  \n");
printf("n=  %d",n);      /* read in the number of nets */
printf("  \n");


for(i = 1;i< 2;i++){
limit = n + 1;
row = n - 1;              /* initialize row and columns */
col = n + 1;
rbound = col + 1;
count = 1;
}



for(i= 0 ; i<=399;i++){
istore[i] = 0;
}









k = 0;
for(i=0;i<limit*2;i+=2){
j = 6 + i;
k = k + 1;
top[k] = save[j];
bottom[k] = save[j+1];          /* read in top and bottom net numbers */
}                               /* from a data file                   */
top[0] = save[2];
bottom[0] = save[3];
top[limit]= save[4];
bottom[limit] = save[5];

MAX = save[j];
keep = j+1;

printf("THE CHANNEL DENSITY SETTING IS CURRENTLY AT");
printf("  \n");
printf("MAX = %d",MAX);
```

```
printf("  \n");
printf("  \n");


anscheck = 1;
printf("WOULD YOU LIKE TO CHANGE CHANNEL DENSITY?:      ");
scanf("%s",answer);
yes = "yes";
anscheck = strcmp(answer,yes);


if(anscheck == 0)
printf("ENTER YOUR NEW DENSITY VALUE:     "),
scanf("%d",&MAX),
printf(" \n"),
printf("THE DENSITY IS NOW SET AT = %d",MAX),
printf(" \n");


k = 0;
for(i=0;i<limit*2;i+=2){          /* read in netnums */
j = 6 + i;                        /* to the map region */
k = k + 1;
MAP_REG[MAX + 1][k].subreg[7].netnum = top[k];
MAP_REG[MAX + 1][k].subreg[1].netnum = top[k];
tkeep[k] = top[k];
MAP_REG[0][k].subreg[1].netnum = bottom[k];
MAP_REG[0][k].subreg[7].netnum = bottom[k];
bkeep[k] = bottom[k];
}




for(i=0;i<=400000;i++){      /* delay program for viewer */
z = z + 1;
}



printf("  \n");
printf("THE PIN NETS LOOK LIKE SO ");
printf("  \n");

for(i=0;i<=n+1;i++){
printf("top%d",i);
printf("= %d",top[i]);
printf("     bottom%d",i);
printf("= %d",bottom[i]);
printf("  \n");
}

for(i=1;i<col;i++){
row = MAX + 1;
```

```
tcol = i;
check = save[keep];
 if(check == 0)
   MAP_REG[row][tcol].subreg[1].layer = "metal1",
   k = 1,
   MAP_REG[row][tcol].subreg[7].layer = "metal1";

                          /* check pin layer type  */

    else if(check != 0)
        MAP_REG[row][tcol].subreg[2].layer = "metal2",
        k = 2,
        MAP_REG[row][tcol].subreg[7].layer = "metal2";



/*********************************************/
/*  DIMENSION THE MAP REGION                 */
/*  Y DIMENSION IS 7 UNITS MINIMUM           */
/*  FOR A CHANNEL AND 19 FOR A               */
/*  SWITCHBOX.                               */
/*                                           */
/*********************************************/


        dimension = 12;
MAP_REG[row][tcol].subreg[k].llx = (tcol - 1)*16;
MAP_REG[row][tcol].subreg[k].lly = MAX * dimension;
MAP_REG[row][tcol].subreg[k].urx =(tcol - 1)*16 + 3;
MAP_REG[row][tcol].subreg[k].ury =MAX * dimension + 3;

MAP_REG[row][tcol].subreg[7].llx = (tcol - 1)*16;
MAP_REG[row][tcol].subreg[7].lly = MAX * dimension;
MAP_REG[row][tcol].subreg[7].urx = MAP_REG[row][tcol].subreg[7].llx;
MAP_REG[row][tcol].subreg[7].ury = MAP_REG[row][tcol].subreg[7].lly;



keep = keep +11;
}
for(i=1;i<col;i++){
row = 0;
tcol = i;
check = save[keep];
 if(check == 0)
   MAP_REG[row][tcol].subreg[1].layer = "metal1",
   k = 1,
   MAP_REG[row][tcol].subreg[7].layer = "metal1";

    else if(check != 0)
        MAP_REG[row][tcol].subreg[2].layer = "metal2",
        k = 2,
        MAP_REG[row][tcol].subreg[7].layer = "metal2";
```

```
 MAP_REG[row][tcol].subreg[k].llx = (tcol - 1)*16;
 MAP_REG[row][tcol].subreg[k].lly = -7;
 MAP_REG[row][tcol].subreg[k].urx = (tcol - 1)*16 + 3;
 MAP_REG[row][tcol].subreg[k].ury = -4;

 MAP_REG[row][tcol].subreg[7].llx = (tcol - 1)*16;
 MAP_REG[row][tcol].subreg[7].lly = -7;
MAP_REG[row][tcol].subreg[7].urx = MAP_REG[row][tcol].subreg[7].llx;
MAP_REG[row][tcol].subreg[7].ury = MAP_REG[row][tcol].subreg[7].lly;


keep = keep +11;
}
```

```
      /*****************************************************************/
      /*                                                               */
      /*                                                               */
      /*  input initial routing area to a magic                        */
      /*  file called temp.mag                                         */
      /*                                                               */
      /*****************************************************************/


in(col,MAX);
```

```
/*****************************************************************************************/
/*                                                                                       */
/*                                                                                       */
/* THIS SECTION WILL FIND THE CORRECT COLUMN TO START AT TO INITIATE                      */
/* AN OUTWARD SCANNING                                                                   */
/* TECHNIQUE BEGINNING FROM A COLUMN WITH MAXIMUM                                        */
/* DENSITY.                                                                              */
/*                                                                                       */
/*                                                                                       */
/*****************************************************************************************/
scol[1] = 0;
start = 0;
Tracks[0] = 0;
```

```
for(i=1;i<=col;i++){
Tracks[i] = number(i,&top[0],&bottom[0],limit);
if(Tracks[i] <= scol[i])
 scol[i+1] = scol[i];
else if(Tracks[i] > scol[i])
 scol[i+1] = Tracks[i],
   keep = i;
}

start = keep;



printf(" \n");
printf("STARTING SCAN IN COLUMN# = %d",start);
printf(" \n");


anscheck = 1;
printf("WOULD YOU LIKE TO CHANGE THE START VALUE?:      ");
scanf("%s",answer);
yes = "yes";
anscheck = strcmp(answer,yes);


if(anscheck == 0)
printf("ENTER YOUR NEW START VALUE:     "),
scanf("%2d",&start),

printf(" \n"),
printf("NOW STARTING SCAN IN COLUMN# = %d",start),
printf(" \n");




count = start;
tcol = count;


/*****************************************************************************/
/*                                                                           */
/*                                                                           */
/* START SCANNING COLUMN BY COLUMN AND APPLY A GREEDY BASED ALGORITHM*/
/* WITH A MODIFIED SCANNING TECHNIQUE. THAT IS THE SCAN WILL GO      */
/* MAX DENSITY TO LEFT AND THEN MAX DENSITY TO RIGHT.                */
/*                                                                           */
/*                                                                           */
/*****************************************************************************/
```

```
                                /*********************************/
                                /*                               */
                                /*                               */
while(count!=0){                /* STARTING LEFT SCAN FIRST      */
                                /*                               */
                                /*                               */
                                /*********************************/

row = MAX +1 ;
trow = row -1;


/*****************************************************/
/*                                                   */
/*                                                   */
/* start scan from column with max density           */
/* only do this section row = max + 1                */
/*                                                   */
/*                                                   */
/*                                                   */
/*****************************************************/


/*****************************/
/*                           */
/* find out what layer       */
/* we have                   */
/*                           */
/*****************************/


check1 = strcmp(MAP_REG[row][tcol].subreg[1].layer,metal1);


if (check1 == 0)
        k=1;
else if (check1 != 0)
        k=2;









/*****************************************************************************/
/*                                                                           */
/*                                                                           */
/* call update to handle                                                     */
/* situation where adjacent net # is zero                                    */
/* or equal.                                                                 */
/*                                                                           */
/*****************************************************************************/
```

```
      /*****************************/
     /*                           */
    /*                           */
   /* update coordinates        */
  /*                           */
 /*                           */
/*************************/

if(MAP_REG[row][tcol].subreg[1].netnum != 0)
update12a(k,trow,tcol,row,MAX,col,dimension);



count = count -1;
tcol = tcol -1;


/*************************************/
/*                                   */
/*                                   */
/*END WHILE LOOP FOR LEFT SCAN       */
/*OF TOP NETS                        */
/*                                   */
/*************************************/


}









/*************************************/
/*                                   */
/*START                              */
/*WHILE LOOP FOR LEFT SCAN           */
/*OF BOTTOM NETS                     */
/*                                   */
/*************************************/


count = start;
tcol = count;
while(count != 0){
row = 0;
trow = row + 1;

check1 = strcmp(MAP_REG[row][tcol].subreg[1].layer,metal1);
```

```
if (check1 == 0)
        k=1;
else if (check1 != 0)
        k=2;

netnum = 0;
for(i = 1;i<= MAX;i++){
netnum = MAP_REG[i][tcol].subreg[1].netnum;

if(netnum == bkeep[tcol])
store[tcol] = 1,

/*****************************************************************************/
/*                                                                           */
/*                                                                           */
/* call update to handle                                                     */
/* situation  where one of the already                                       */
/* occupied tracks have the same net number                                  */
/* pin entrance is from the bottom of routing region                         */
/*****************************************************************************/


update4a4b(k,trow,tcol,row,MAX,i,dimension);
} /* end for loop */



count = count - 1;




if(count == 0){    /* check all bottom pins that are still not in the */
                   /* routing region                                  */

for(j = 1;j<= start;j++){



netnum = bkeep[j];


if(store[j] != 1){
check2 = 0;
for(i = 1;i<= MAX;i++){
check = MAP_REG[i][j].subreg[1].netnum;

if(check == 0)
store[j] = 1,
trow = j + 1,
```

```
/*************************************************************************/
/*                                                                       */
/* call update to handle                                                 */
/* situation  where none of the already                                  */
/* occupied tracks have the same net number                              */
/* and a new track must be occupied .                                    */
/* pin entrance is from the bottom of routing region                     */
/*************************************************************************/

update5a5b(k,trow,j,row,MAX,i,dimension),
check2 = 1,
i = MAX + 1;

} /* end second for loop */

if(check2 != 1){      /* last check for a free track */
  check1 = 0;
  for(l = 1;l<=MAX;l++){
   test = MAP_REG[l][j].subreg[1].netnum;

    if(test == netnum)
     check1 = 1,
     update4a4b(k,trow,j,row,MAX,l,dimension),
     l = MAX + 1;
 } /* end for loop l */
  if(check1 == 0)  /* we need to increase channel density */
      message(l);
} /* end if check2 != 1  */


} /* end second if       */
} /* end first for loop  */
} /* end first if        */




tcol = tcol - 1;


   /*************************/
   /*                       */
   /*                       */
}  /* end while bottom scan */
   /*                       */
   /*                       */
   /*************************/
```

```
/*****************************************************************************/

count = start + 1;
tcol = count;
                                /*********************************/
                                /*                               */
                                /*                               */
while(count != col + 1){        /* STARTING RIGHT SCAN           */
                                /*                               */
                                /*                               */
                                /*********************************/
row = MAX +1 ;
trow = row -1;
netnum = tkeep[tcol];


/*****************************************************/
/*                                                   */
/*                                                   */
/* start scan from column with max density           */
/* only do this section row = max + 1                */
/*                                                   */
/*                                                   */
/*                                                   */
/*****************************************************/


/*****************************/
/*                           */
/* find out what layer       */
/* we have                   */
/*                           */
/*****************************/


check1 = strcmp(MAP_REG[row][tcol].subreg[1].layer,metal1);


if (check1 == 0)
        k=1;
else if (check1 != 0)
        k=2;
```

```
/*************************************************************************/
/*                                                                       */
/*                                                                       */
/* call update to handle                                                 */
/* situation where we first we look for a track                          */
/* with the same net number                                              */
/*                                                                       */
/*************************************************************************/

check = 0;
z = MAX;
for(j = 1; j<=MAX; j++){
z = z - 1;
if(netnum == MAP_REG[j][tcol].subreg[1].netnum && netnum != 0)
     check = 1,

       /*****************************/
      /*                           */
     /*                           */
    /* update coordinates        */
   /*                           */
  /*                           */
 /*****************************/

updater12a(k,dimension,z,row,tcol),
j = MAX + 1;

[ /* end for loop for same net number */



/*************************************************/
/*                                               */
/*                                               */
/*                                               */
/*NOW WE MUST HANDLE SITUATION WHERE A           */
/*TOP NET IN THE RIGHT SCAN AREA IS              */
/*ENTERING THE ROUTING REGION FOR THE            */
/*FIRST TIME. THEREFORE WE MUST CHECK            */
/*THE BOTTOM NETS FROM START PLUS ONE            */
/*TO SEE IF WE MUST EXTEND TRACK IN THE          */
/*LEFT SCAN DIRECTION AND OF COURSE              */
/*CHECK AHEAD INTO THE RIGHT SCAN AREA.          */
/*                                               */
/*************************************************/
```

```
                    /*****************************************/
                    /*                                       */
if(check == 0){     /* then we check for a free track */
                    /*                                       */
                    /*****************************************/

for(j = 1;j<=MAX;j++){
if(MAP_REG[j][tcol].subreg[1].netnum == 0 && netnum != 0)
     check = 1,
   updater3a3b(k,trow,tcol,dimension,j,row,col),
     j = MAX + 1;
}  /* end for loop j */
}  /* end if check  */
```

```
count = count + 1;
tcol = tcol + 1;
```

```
     /*************************************/
    /*                                   */
} /* end while loop right scan */
 /*  top nets.                         */
/*************************************/
```

```
/*****************************************/
/*                                       */
/*  START                                */
/*  WHILE LOOP FOR RIGHT  SCAN */
/*  OF BOTTOM NETS                       */
/*                                       */
/*****************************************/
```

```
count = start+1;
tcol = count;
while(count != col + 1){
row = 0;
trow = row + 1;

check1 = strcmp(MAP_REG[row][tcol].subreg[1].layer,metal1);


if (check1 == 0)
        k=1;
else if (check1 != 0)
        k=2;

netnum = 0;
for(i = 1;i<= MAX;i++){
netnum = MAP_REG[i][tcol].subreg[1].netnum;

if(netnum == bkeep[tcol])
store[tcol] = 1,

/*****************************************************************************/
/*                                                                           */
/*                                                                           */
/* call update to handle                                                     */
/* situation where one of the already                                        */
/* occupied tracks have the same net number                                  */
/* pin entrance is from the bottom of routing region                         */
/*****************************************************************************/


update4a4b(k,trow,tcol,row,MAX,i,dimension);
} /* end for loop */


count = count + 1;




/* check all bottom pins that are still not in the*/
/*              routing region                    */
if(count == col + 1){

for(j = start + 1;j<= col;j++){
```

```
for(z=1; z<=MAX; z++){
netnum = MAP_REG[z][j].subreg[1].netnum;

if(netnum == bkeep[j] && store[j] != 1)
store[j] = 1,




/*****************************************************************************/
/*                                                                           */
/*                                                                           */
/* call update to handle                                                     */
/* situation where one of the already                                        */
/* occupied tracks have the same net number                                  */
/* pin entrance is from the bottom of routing region                         */
/*****************************************************************************/


update4a4b(k,trow,j,row,MAX,z,dimension),
z = MAX + 1;
} /* end for loop k */









if(store[j] != 1){
store[j] = 1;
/*****************************************************************************/
/*                                                                           */
/* call update to handle                                                     */
/* situation where none of the already                                       */
/* occupied tracks have the same net number                                  */
/* and a new track must be occupied .                                        */
/* pin entrance is from the bottom of routing region                         */
/*****************************************************************************/

update5a5b(k,trow,j,row,MAX,i,dimension);
} /* end  if store  */
} /* end j <= col for loop */

} /* end if count = col + 1 */




tcol = tcol + 1;
```

```
    /*******************************/
    /*                             */
    /*                             */
}   /* end while bottom scan */
    /*                             */
    /*                             */
    /*******************************/
```

```
/*****************************************************************************************/
/*                                                                                       */
/*                                                                                       */
/*  THIS SECTION OUTPUTS THE CHANNEL ROUTING AREA TO A FILE                              */
/*  OF TYPE MAGIC                                                                        */
/*                                                                                       */
/*                                                                                       */
/*                                                                                       */
/*****************************************************************************************/

out(MAX,col);
```

```
/*****************************************************************************/
/*                                                                           */
/*                                                                           */
/*  THIS SECTION ALLOWS THE USER TO CREATE A NEW DATA FILE TO                */
/*  DEFINE A NEW ROUTING PROBLEM. THE USER CAN THEN USE THIS                 */
/*  DATA FILE AS INPUT TO THE ROUTER.                                        */
/*                                                                           */
/*                                                                           */
/*****************************************************************************/
anscheck = 1;
printf("WOULD YOU LIKE TO MAKE A NEW ROUTING DATA FILE?:      ");
scanf("%s",answer);
yes = "yes";
anscheck = strcmp(answer,yes);


if(anscheck == 0)
 input(0),
 printf("THE ROUTING IS OVER ___ BYE!!!!!!!!");
else if (anscheck != 0)
 printf("THE ROUTING IS OVER ___ BYE!!!!!!!!");


printf("     \n");
```

```
 /*****************************************************************************/
 /*                                                                           */
 /*                                                                           */
 /*                                                                           */
 /*                                                                           */
}/*          THIS IS THE END OF MAIN  ------  ROUTER.C                        */
 /*                                                                           */
 /*                                                                           */
 /*                                                                           */
 /*                                                                           */
 /*                                                                           */
 /*****************************************************************************/
```

```
/*****************************************************************************/
/*                                                                           */
/*                                                                           */
/*  THIS ROUTINE IS USED FOR READING THE INPUT DATA FILE.                    */
/*  THE ROUTINE CAN DIFFERENTIATE BETWEEN A CHARACTER STRING                 */
/*  AND A NUMBER.                                                            */
/*                                                                           */
/*                                                                           */
/*****************************************************************************/


nextline(fp)
FILE *fp;
{
      if(fgets(buf,255,fp) == NULL)
         return(0);

      if (isdigit(*buf) || (*buf == '-')) -
        sscanf(buf,"%d",&num);
          return(2);
        }

       buf[strlen(buf)-1] = ' 0';
       return(1);
}
```

IN.C

```
#include "globe.h"
#include <stdio.h>


PROC in(col,MAX)
int col,MAX;
{
int i,l,k;
FILE *fp,*fopen();




/*****************************************************************************/
/*                                                                           */
/*                                                                           */
/*  THIS SECTION OUTPUTS THE INITIAL CHANNEL ROUTING AREA TO A FILE          */
/*  OF TYPE MAGIC                                                            */
/*                                                                           */
/*                                                                           */
/*                                                                           */
/*****************************************************************************/


fp = fopen("temp.mag","w");
fprintf(fp,"%s\n","magic");
fprintf(fp,"%s %s\n","tech","scmos");

fprintf(fp,"%s %s\n","timestamp","515112872");



fprintf(fp,"%s %s %s\n","<<","metal1",">>");
k=1;
row=0;
  for(i=1;i<col;i++){
   tcol=i;
  if(MAP_REG[row][tcol].subreg[k].llx!=0 ||
     MAP_REG[row][tcol].subreg[k].urx!=0)
fprintf(fp,"%s %d %d %d %d\"rect",MAP_REG[row][tcol].subreg[k].llx,
   MAP_REG[row][tcol].subreg[k].lly,
   MAP_REG[row][tcol].subreg[k].urx,
   MAP_REG[row][tcol].subreg[k].ury);
}
row=1+MAX;
  for(i=1;i<col;i++){
   tcol=i;
  if(MAP_REG[row][tcol].subreg[k].llx!=0 ||
  MAP_REG[row][tcol].subreg[k].urx!=0)
fprintf(fp,"%s %d %d %d %d\n","rect",MAP_REG[row][tcol].subreg[k].llx,
   MAP_REG[row][tcol].subreg[k].lly,
   MAP_REG[row][tcol].subreg[k].urx,
   MAP_REG[row][tcol].subreg[k].ury);
```

```
}
fprintf(fp,"%s %s %s\n",<<","metal2",">>");
k=2;
row=0;
  for(i=1;i<col;i++){
   tcol=i;
  if(MAP_REG[row][tcol].subreg[k].llx!=0 ||
     MAP_REG[row][tcol].subreg[k].urx!=0)
fprintf(fp,"%s %d %d %d %d\n","rect",MAP_REG[row][tcol].subreg[k].llx,
   MAP_REG[row][tcol].subreg[k].lly,
   MAP_REG[row][tcol].subreg[k].urx,
   MAP_REG[row][tcol].subreg[k].ury);
}
row=1+MAX;
  for(i=1;i<col;i++){
   tcol=i;
  if(MAP_REG[row][tcol].subreg[k].llx!=0 ||
     MAP_REG[row][tcol].subreg[k].urx!=0)
fprintf(fp,"%s %d %d %d %d\n","rect",MAP_REG[row][tcol].subreg[k].llx,
   MAP_REG[row][tcol].subreg[k].lly,
   MAP_REG[row][tcol].subreg[k].urx,
   MAP_REG[row][tcol].subreg[k].ury);
}



fprintf(fp,"%s %s %s\n","<<","labels",">>");

l = 7;
row =0;
  for(i=1;i<col;i++){
   tcol=i;
   fprintf(fp,"%s %s %d %d %d %d %d %d\n","rlabel",
   MAP_REG[row][tcol].subreg[l].layer,
   MAP_REG[row][tcol].subreg[l].llx,
   MAP_REG[row][tcol].subreg[l].lly,
   MAP_REG[row][tcol].subreg[l].urx,
   MAP_REG[row][tcol].subreg[l].ury,1,
   MAP_REG[row][tcol].subreg[l].netnum);
}

l = 7;
row = MAX +1;
  for(i=1;i<col;i++){
   tcol=i;
   fprintf(fp,"%s %s %d %d %d %d %d %d\n","rlabel",
   MAP_REG[row][tcol].subreg[l].layer,
   MAP_REG[row][tcol].subreg[l].llx,
   MAP_REG[row][tcol].subreg[l].lly,
   MAP_REG[row][tcol].subreg[l].urx,
   MAP_REG[row][tcol].subreg[l].ury,1,
```

```
   MAP_REG[row][tcol].subreg[1].netnum);
}


fprintf(fp,"%s %s %s\n","<<","end",">>");
fclose(fp);



return(0);

}
```

OUT.C

```
#include "globe.h"
#include <stdio.h>



/*****************************************************************************/
/*                                                                           */
/*  THIS SECTION OUTPUTS THE CHANNEL ROUTING AREA TO A FILE                  */
/*  OF TYPE MAGIC                                                            */
/*                                                                           */
/*                                                                           */
/*                                                                           */
/*****************************************************************************/
PROC out(MAX,col)
int MAX,col;
{
int i,l,k;
FILE *fp,*fopen();
fp = fopen("route.mag","w");
fprintf(fp,"%s\n","magic");
fprintf(fp,"%s %s\n","tech","scmos");
fprintf(fp,"%s %s\n","timestamp","515112872");



fprintf(fp,"%s %s %s\n","<<","metal1",">>");
k=1;
for(row =0;row<=MAX + 1;row++)
  for(i=1;i<col;i++){
   tcol=i;
if(MAP_REG[row][tcol].subreg[k].llx !=0 ||
   MAP_REG[row][tcol].subreg[k].urx !=0)
fprintf(fp,"%s %d %d %d %d\n","rect",MAP_REG[row][tcol].subreg[k].llx,
   MAP_REG[row][tcol].subreg[k].lly,
   MAP_REG[row][tcol].subreg[k].urx,
   MAP_REG[row][tcol].subreg[k].ury);
}



k=19;
for(row =0;row<=MAX + 1;row++)
  for(i=1;i<col;i++){
   tcol=i;
if(MAP_REG[row][tcol].subreg[k].llx !=0 ||
   MAP_REG[row][tcol].subreg[k].urx !=0)
fprintf(fp,"%s %d %d %d %d\n","rect",MAP_REG[row][tcol].subreg[k].llx,
   MAP_REG[row][tcol].subreg[k].lly,
   MAP_REG[row][tcol].subreg[k].urx,
   MAP_REG[row][tcol].subreg[k].ury);
}
```

```
fprintf(fp,"%s %s %s n","<<","metal2",">>");
k=2;
for(row =0;row<=MAX + 1;row++)
  for(i=1;i<col;i++){
   tcol=i;
if(MAP_REG[row][tcol].subreg[k].llx !=0 ||
   MAP_REG[row][tcol].subreg[k].urx !=0)
fprintf(fp,"%s %d %d %d %d\n","rect",MAP_REG[row][tcol].subreg[k].llx,
   MAP_REG[row][tcol].subreg[k].lly,
   MAP_REG[row][tcol].subreg[k].urx,
   MAP_REG[row][tcol].subreg[k].ury);
}
k=5;
for(row =0;row<=MAX + 1;row++)
  for(i=1;i<col;i++){
   tcol=i;
if(MAP_REG[row][tcol].subreg[k].llx !=0 ||
   MAP_REG[row][tcol].subreg[k].urx !=0)
fprintf(fp,"%s %d %d %d %d\n","rect",MAP_REG[row][tcol].subreg[k].llx,
   MAP_REG[row][tcol].subreg[k].lly,
   MAP_REG[row][tcol].subreg[k].urx,
   MAP_REG[row][tcol].subreg[k].ury);
}



fprintf(fp,"%s %s %s\n","<<","m2contact",">>");
for (k=3;k<=4;k++)
for(row =0;row<=MAX + 1;row++)
  for(i=1;i<col;i++){
   tcol=i;
if(MAP_REG[row][tcol].subreg[k].llx !=0 ||
   MAP_REG[row][tcol].subreg[k].urx !=0)
fprintf(fp,"%s %d %d %d %d\n","rect",MAP_REG[row][tcol].subreg[k].llx,
   MAP_REG[row][tcol].subreg[k].lly,
   MAP_REG[row][tcol].subreg[k].urx,
   MAP_REG[row][tcol].subreg[k].ury);
}

k = 8;
for(row =0;row<=MAX + 1;row++)
  for(i=1;i<col;i++){
   tcol=i;
if(MAP_REG[row][tcol].subreg[k].llx !=0 ||
   MAP_REG[row][tcol].subreg[k].urx !=0)
fprintf(fp,"%s %d %d %d %d\n","rect",MAP_REG[row][tcol].subreg[k].llx,
   MAP_REG[row][tcol].subreg[k].lly,
   MAP_REG[row][tcol].subreg[k].urx,
   MAP_REG[row][tcol].subreg[k].ury);
}

k = 6;
for(row =0;row<=MAX + 1;row++)
  for(i=1;i<col;i++){
```

```
   tcol=i;
if(MAP_REG[row][tcol].subreg[k].llx !=0 ||
   MAP_REG[row][tcol].subreg[k].urx !=0)
fprintf(fp,"%s %d %d %d %d\n","rect",MAP_REG[row][tcol].subreg[k].llx,
   MAP_REG[row][tcol].subreg[k].lly,
   MAP_REG[row][tcol].subreg[k].urx,
   MAP_REG[row][tcol].subreg[k].ury);
}



fprintf(fp,"%s %s %s\n","<<","labels",">>");

l = 7;
row =0;
  for(i=1;i<col;i++){
   tcol=i;
   fprintf(fp,"%s %s %d %d %d %d %d %d\n","rlabel",
   MAP_REG[row][tcol].subreg[l].layer,
   MAP_REG[row][tcol].subreg[l].llx,
   MAP_REG[row][tcol].subreg[l].lly,
   MAP_REG[row][tcol].subreg[l].urx,
   MAP_REG[row][tcol].subreg[l].ury,1,
   MAP_REG[row][tcol].subreg[l].netnum);
}

l = 7;
row = MAX +1;
  for(i=1;i<col;i++){
   tcol=i;
   fprintf(fp,"%s %s %d %d %d %d %d %d\n","rlabel",
   MAP_REG[row][tcol].subreg[l].layer,
   MAP_REG[row][tcol].subreg[l].llx,
   MAP_REG[row][tcol].subreg[l].lly,
   MAP_REG[row][tcol].subreg[l].urx,
   MAP_REG[row][tcol].subreg[l].ury,1,
   MAP_REG[row][tcol].subreg[l].netnum);
}

fprintf(fp,"%s %s %s\n","<<","end",">>");
fclose(fp);



return(0);
```

```
/*****************************************************************************/
/*                                                                           */
/*                                                                           */
/*                                                                           */
/*                                                                           */
}/*              THIS IS THE END OF OUT                                       */
/*                                                                           */
/*                                                                           */
/*                                                                           */
/*                                                                           */
/*****************************************************************************/
```

UPDATE12A.C

```
#include "globe.h"
#include <stdio.h>


PROC update12a(k,trow,tcol,row,MAX,col,dimension)
int k,trow,tcol,row,MAX,col,dimension;
{


 /*************************************************************************/
 /*                                                                       */
 /*                                                                       */
 /*                                                                       */
 /*   ROUTINE TO HANDLE COORDINATE UPDATE FOR                             */
 /*                                                                       */
 /*   LEFT SCAN REGION ONLY .THIS IS THE SITUATION                        */
 /*                                                                       */
 /*   WHERE ENTERING NETNUM IS METAL1 or METAL2                           */
 /*                                                                       */
 /*   AND ADJACENT GRID IS ZERO.ENTRANCE IS FROM TOP                      */
 /*                                                                       */
 /*   OF CHANNEL.                                                         */
 /*                                                                       */
 /*************************************************************************/

int sub1,savecol,j,i,netnum,srow,dim0,dim1,ydim;
int sub1a,sub3,sub2,fact,saverow,savritecol,f;

dim1 = dimension - 8;
dim0 = dimension - 4;
srow = 0;
 sub1 = 0;
 sub1a = 0;
 sub2 = 0;
 sub3 = 0;
 savecol = 0;
savritecol = 0;
f = 0;
fact = 0;

netnum =  MAP_REG[MAX+1][tcol].subreg[1].netnum;
i = tcol - 1;
while(i != 0){
if(tkeep[i] == netnum || bkeep[i] == netnum)
 savecol = i,
fact = (tcol - savecol)*16;

i = i - 1;
}
```

```
i = tcol + 1;
while(i <= col){
if(tkeep[i] == netnum || bkeep[i] == netnum)
 savritecol = i,
f = (savritecol - tcol)*16;

i = i + 1;
}
```

```
if (MAP_REG[trow][tcol].subreg[1].netnum == 0 ||
    MAP_REG[trow][tcol].subreg[1].netnum ==
    MAP_REG[row][tcol].subreg[1].netnum){
```

```
 if (k == 1)
MAP_REG[trow][tcol].subreg[3].urx=MAP_REG[row][tcol].subreg[1].llx+4,
MAP_REG[trow][tcol].subreg[3].ury=MAP_REG[row][tcol].subreg[1].lly,
MAP_REG[trow][tcol].subreg[3].llx=MAP_REG[row][tcol].subreg[1].llx,
MAP_REG[trow][tcol].subreg[3].lly=MAP_REG[row][tcol].subreg[1].lly-4,
 sub1 = 1;

 else if (k == 2)
MAP_REG[trow][tcol].subreg[2].urx=MAP_REG[row][tcol].subreg[2].llx+3,
MAP_REG[trow][tcol].subreg[2].ury=MAP_REG[row][tcol].subreg[2].lly,
MAP_REG[trow][tcol].subreg[2].llx=MAP_REG[row][tcol].subreg[2].llx,
MAP_REG[trow][tcol].subreg[2].lly=MAP_REG[row][tcol].subreg[2].lly-dim0,
 sub2 = 1;
```

```
 if (sub1 == 1)
MAP_REG[trow][tcol].subreg[2].urx=MAP_REG[trow][tcol].subreg[3].llx+3,
MAP_REG[trow][tcol].subreg[2].ury=MAP_REG[trow][tcol].subreg[3].lly,
MAP_REG[trow][tcol].subreg[2].llx=MAP_REG[trow][tcol].subreg[3].llx,
MAP_REG[trow][tcol].subreg[2].lly=MAP_REG[trow][tcol].subreg[3].lly-dim1,
 sub1a = 1;

 else if (sub2 == 1)
MAP_REG[trow][tcol].subreg[4].urx=MAP_REG[trow][tcol].subreg[2].llx+4,
MAP_REG[trow][tcol].subreg[4].ury=MAP_REG[trow][tcol].subreg[2].lly,
MAP_REG[trow][tcol].subreg[4].llx=MAP_REG[trow][tcol].subreg[2].llx,
MAP_REG[trow][tcol].subreg[4].lly=MAP_REG[trow][tcol].subreg[2].lly-4,
 sub3 = 1;
```

```
   if (sub1a == 1)
MAP_REG[trow][tcol].subreg[4].urx=MAP_REG[trow][tcol].subreg[2].llx+4,
```

```
MAP_REG[trow][tcol].subreg[4].ury=MAP_REG[trow][tcol].subreg[2].lly,
MAP_REG[trow][tcol].subreg[4].llx=MAP_REG[trow][tcol].subreg[2].llx,
MAP_REG[trow][tcol].subreg[4].lly=MAP_REG[trow][tcol].subreg[2].lly-4,
   sub3 = 1;

  if (sub3 == 1)
MAP_REG[trow][tcol].subreg[19].urx=MAP_REG[trow][tcol].subreg[4].llx,
MAP_REG[trow][tcol].subreg[19].ury=MAP_REG[trow][tcol].subreg[4].lly+3,
MAP_REG[trow][tcol].subreg[19].llx=
MAP_REG[trow][tcol].subreg[4].llx -fact,
MAP_REG[trow][tcol].subreg[19].lly=MAP_REG[trow][tcol].subreg[4].lly;

  if (savritecol != 0)
MAP_REG[trow][tcol].subreg[19].urx=
MAP_REG[trow][tcol].subreg[19].urx+4+f,
MAP_REG[trow][tcol].subreg[19].ury=MAP_REG[trow][tcol].subreg[19].ury,
MAP_REG[trow][tcol].subreg[19].llx=MAP_REG[trow][tcol].subreg[19].llx,
MAP_REG[trow][tcol].subreg[19].lly=MAP_REG[trow][tcol].subreg[19].lly;

  if (savritecol == 0)
MAP_REG[trow][tcol].subreg[19].urx=MAP_REG[trow][tcol].subreg[19].urx+12,
MAP_REG[trow][tcol].subreg[19].ury=MAP_REG[trow][tcol].subreg[19].ury,
MAP_REG[trow][tcol].subreg[19].llx=MAP_REG[trow][tcol].subreg[19].llx,
MAP_REG[trow][tcol].subreg[19].lly=MAP_REG[trow][tcol].subreg[19].lly;


}


if (MAP_REG[trow][tcol].subreg[1].netnum != 0 ||
    MAP_REG[trow][tcol].subreg[1].netnum !=
    MAP_REG[row][tcol].subreg[1].netnum && sub3 != 1){



/*****************************************/
/*                                       */
/* CALL UPDATE3A3B.C IF PIN COULD        */
/* NOT ENTER ROUTING                     */
/* REGION                                */
/*                                       */
/*****************************************/

if(sub3 != 1 )
ydim = dimension,
srow=
update3a3b(k,trow,tcol,row,MAX,savecol,fact,savritecol,f,netnum,ydim);



}
```

```
if(sub3 == 1 && srow == 0){
for(i = 0;i <= tcol;i++){
MAP_REG[trow][i].subreg[1].netnum = netnum;  /* update track with */
}                                            /* appropriate netnumber */
}


if(savritecol != 0 && srow == 0){
for(i = 0;i <= savritecol;i++){
MAP_REG[trow][i].subreg[1].netnum = netnum;  /* update track with */
}                                            /* appropriate netnumber */
}




return(k);
}
```

UPDATE3A3B.C

```
/*****************************************************************************/
/*                                                                           */
/*                                                                           */
/* ROUTINE TO HANDLE SITUATION WHERE PIN IS METAL1 OR                        */
/* METAL2 ENTERING FROM TOP OF ROUTING REGION                                */
/* BUT ADJACENT NETNUM IS NOT EQUAL.                                         */
/*                                                                           */
/*****************************************************************************/


#include "globe.h"
#include <stdio.h>
PROC update3a3b(k,trow,tcol,row,
                MAX,savecol,fact,savritecol,f,netnum,dimension)
int k,trow,tcol,row,MAX,savecol,fact,savritecol,f,netnum,dimension;
{
int save,tcount,dense,srow,j,i,dim0,dim1,ydim;
int sub3a,sub3b,sub3c,sub3d,0;
   dim0 = dimension -4;
   dim1 = dimension -8;
  ydim = dimension;


    nrow = trow -1;      /* increment to the next row and check */
   save = trow;
    0 = 1;              /* set row offset counter */

         sub3a = 0,
         sub3b = 0,
         sub3c = 0,
         sub3d = 0;
         tcount = 1;
         save = 0;
         srow = 0;


   while(nrow != 0){

  if(MAP_REG[nrow][tcol].subreg[1].netnum == 0 ||
    MAP_REG[nrow][tcol].subreg[1].netnum ==
    MAP_REG[row][tcol].subreg[1].netnum){


   if (k == 1)
MAP_REG[trow][tcol].subreg[3].urx=MAP_REG[row][tcol].subreg[1].llx+4,
MAP_REG[trow][tcol].subreg[3].ury=MAP_REG[row][tcol].subreg[1].lly,
MAP_REG[trow][tcol].subreg[3].llx=MAP_REG[row][tcol].subreg[1].llx,
MAP_REG[trow][tcol].subreg[3].lly=MAP_REG[row][tcol].subreg[1].lly-4,
   sub3a = 1;

   else if (k == 2)
MAP_REG[trow][tcol].subreg[2].urx=MAP_REG[row][tcol].subreg[2].llx+3,
MAP_REG[trow][tcol].subreg[2].ury=MAP_REG[row][tcol].subreg[2].lly,
MAP_REG[trow][tcol].subreg[2].llx=MAP_REG[row][tcol].subreg[2].llx,
MAP_REG[trow][tcol].subreg[2].lly=MAP_REG[row][tcol].subreg[2].lly-dim0,
```

```
   sub3b = 1;


   if (sub3a== 1)
MAP_REG[trow][tcol].subreg[2].urx=MAP_REG[trow][tcol].subreg[3].llx+3,
MAP_REG[trow][tcol].subreg[2].ury=MAP_REG[trow][tcol].subreg[3].lly,
MAP_REG[trow][tcol].subreg[2].llx=MAP_REG[trow][tcol].subreg[3].llx,
MAP_REG[trow][tcol].subreg[2].lly=MAP_REG[trow][tcol].subreg[3].lly-dim1,
   sub3b = 1;


   if (sub3b == 1)
MAP_REG[nrow][tcol].subreg[2].urx=MAP_REG[trow][tcol].subreg[2].llx+3,
MAP_REG[nrow][tcol].subreg[2].ury=MAP_REG[trow][tcol].subreg[2].lly,
MAP_REG[nrow][tcol].subreg[2].llx=MAP_REG[trow][tcol].subreg[2].llx,
MAP_REG[nrow][tcol].subreg[2].lly=
MAP_REG[trow][tcol].subreg[2].lly-ydim*0,
   sub3c =1;


   if (sub3c == 1)
MAP_REG[nrow][tcol].subreg[4].urx=MAP_REG[nrow][tcol].subreg[2].llx+4,
MAP_REG[nrow][tcol].subreg[4].ury=MAP_REG[nrow][tcol].subreg[2].lly,
MAP_REG[nrow][tcol].subreg[4].llx=MAP_REG[nrow][tcol].subreg[2].llx,
MAP_REG[nrow][tcol].subreg[4].lly=MAP_REG[nrow][tcol].subreg[2].lly-4,
   sub3d = 1;

   if (sub3d == 1 )
MAP_REG[nrow][tcol].subreg[19].urx=MAP_REG[nrow][tcol].subreg[4].llx,
MAP_REG[nrow][tcol].subreg[19].ury=MAP_REG[nrow][tcol].subreg[4].lly+3,
MAP_REG[nrow][tcol].subreg[19].llx=
MAP_REG[nrow][tcol].subreg[4].llx - fact,
MAP_REG[nrow][tcol].subreg[19].lly=MAP_REG[nrow][tcol].subreg[4].lly,
   srow = nrow,
   nrow = 1;    /* set nrow = 0 you found a good usable track */


   if (savritecol != 0)
MAP_REG[srow][tcol].subreg[19].urx =
MAP_REG[srow][tcol].subreg[19].urx +4+ 8+f,
MAP_REG[srow][tcol].subreg[19].ury=MAP_REG[srow][tcol].subreg[19].ury,
MAP_REG[srow][tcol].subreg[19].llx=MAP_REG[srow][tcol].subreg[19].llx,
MAP_REG[srow][tcol].subreg[19].lly=MAP_REG[srow][tcol].subreg[19].lly;

   if (savritecol == 0)
MAP_REG[srow][tcol].subreg[19].urx=MAP_REG[srow][tcol].subreg[4].urx+8,
MAP_REG[srow][tcol].subreg[19].ury=MAP_REG[srow][tcol].subreg[4].lly+3,
MAP_REG[srow][tcol].subreg[19].llx=MAP_REG[srow][tcol].subreg[4].urx,
MAP_REG[srow][tcol].subreg[19].lly=MAP_REG[srow][tcol].subreg[4].lly;

}


 if (sub3d == 1){
for(i = 0;i<=tcol;i++){
```

```
  MAP_REG[srow][i].subreg[1].netnum = netnum;

}    /* update track with the appropriate netnumber */
}

 if (sub3d == 1 && savritecol != 0){
for(i = 0;i<= savritecol;i++){
  MAP_REG[srow][i].subreg[1].netnum = netnum;

}
}
```

```
O = O + 1;
nrow = nrow -1;
if(tcount == save)
 dense = MAX + 1,
 printf("THE CHANNEL DENSITY CAPACITY HAS BEEN EXCEEDED !!! \n"),
 printf("THEREFORE YOU MUST RESTART THE PROGRAM AND INCREASE\n"),
 printf("THE DENSITY BY ONE\n"),
 printf("DENSITY = %d",dense),
 printf("   \n"),
 printf("START = %d",start),
 printf("   \n");


}

return(srow);
}
```

UPDATE4A4B.C

```
#include "globe.h"
#include <stdio.h>

/*****************************************************************************/
/*                                                                           */
/*PROGRAM HANDLE                                                             */
/*SITUATION OF PIN ENTERING CHANNEL                                          */
/*FROM BOTTOM TO A TRACK ALREADY OCCUPIED                                    */
/*WITH SAME NETNUM.USED IN THE LEFT SCAN AND                                 */
/*RIGHT SCAN AREA.                                                           */
/*****************************************************************************/



PROC update4a4b(k,trow,tcol,row,MAX,j,dimension)

int k,trow,tcol,row,MAX,j,dimension;
{
int check,save,tcount,dense,i,srow,ydim,fact,m,f;
int sub4,sub4a,sub4b,sub4c,sub4d,sub4e,sub4f,sub4g,netnum;
ydim = dimension;
netnum = bkeep[tcol];


    nrow = j;
    save = MAX;

check  =  strcmp(MAP_REG[row][tcol].subreg[1].layer,metal1);
if(check == 0)
 k = 1;
else if(check != 0)
 k = 2;


f = 0;
i = tcol + 1;
while(i <= col){
if(tkeep[i] == netnum)
f = (i - tcol)*16,
i = col + 1;
i = i + 1;
}



sub4 = 0;
sub4a = 0;
sub4b = 0;
sub4c = 0;
sub4d = 0;
sub4e = 0;
sub4f = 0;
sub4g = 0;
row = 0;
```

```
if(netnum != 0 ){
   if (k  == 1)
MAP_REG[row][tcol].subreg[1].urx=MAP_REG[row][tcol].subreg[1].urx+9,
MAP_REG[row][tcol].subreg[1].ury=MAP_REG[row][tcol].subreg[1].ury,
MAP_REG[row][tcol].subreg[1].llx=MAP_REG[row][tcol].subreg[1].llx,
MAP_REG[row][tcol].subreg[1].lly=MAP_REG[row][tcol].subreg[1].lly,
MAP_REG[row][tcol].subreg[4].urx=MAP_REG[row][tcol].subreg[1].urx,
MAP_REG[row][tcol].subreg[4].ury=MAP_REG[row][tcol].subreg[1].ury,
   sub4c =1;


   else if (k == 2)
MAP_REG[row][tcol].subreg[2].urx=MAP_REG[row][tcol].subreg[2].urx+9,
MAP_REG[row][tcol].subreg[2].ury=MAP_REG[row][tcol].subreg[2].ury,
MAP_REG[row][tcol].subreg[2].llx=MAP_REG[row][tcol].subreg[2].llx,
MAP_REG[row][tcol].subreg[2].lly=MAP_REG[row][tcol].subreg[2].lly,
   m = 0,
   sub4a = 1;


   if (sub4c == 1)
MAP_REG[row][tcol].subreg[4].llx=MAP_REG[row][tcol].subreg[1].urx-4,
MAP_REG[row][tcol].subreg[4].lly=MAP_REG[row][tcol].subreg[1].lly-1,
MAP_REG[row][tcol].subreg[2].urx=MAP_REG[row][tcol].subreg[4].urx,
MAP_REG[row][tcol].subreg[2].ury=MAP_REG[row][tcol].subreg[4].ury+4,
MAP_REG[row][tcol].subreg[2].llx=MAP_REG[row][tcol].subreg[4].llx+1,
MAP_REG[row][tcol].subreg[2].lly=MAP_REG[row][tcol].subreg[4].lly+4,
   m = 4,
   sub4a = 1;


 if (sub4a == 1)
   fact = (nrow * ydim) -(ydim - 7) - m,
MAP_REG[nrow][tcol].subreg[5].urx=MAP_REG[row][tcol].subreg[2].urx,
MAP_REG[nrow][tcol].subreg[5].ury=MAP_REG[row][tcol].subreg[2].ury+fact,
MAP_REG[nrow][tcol].subreg[5].llx=MAP_REG[row][tcol].subreg[2].urx-3,
MAP_REG[nrow][tcol].subreg[5].lly=MAP_REG[row][tcol].subreg[2].ury,
   sub4b = 1;


 if (sub4b == 1)
MAP_REG[nrow][tcol].subreg[6].urx=MAP_REG[nrow][tcol].subreg[5].urx,
MAP_REG[nrow][tcol].subreg[6].ury=MAP_REG[nrow][tcol].subreg[5].ury+1,
MAP_REG[nrow][tcol].subreg[6].llx=MAP_REG[nrow][tcol].subreg[5].urx-4,
MAP_REG[nrow][tcol].subreg[6].lly=MAP_REG[nrow][tcol].subreg[5].ury-3,
  sub4c = 1;


 if (sub4c == 1)
MAP_REG[nrow][tcol].subreg[19].urx=MAP_REG[nrow][tcol].subreg[6].llx,
MAP_REG[nrow][tcol].subreg[19].ury=MAP_REG[nrow][tcol].subreg[6].ury-1,
MAP_REG[nrow][tcol].subreg[19].llx=MAP_REG[nrow][tcol].subreg[6].llx-8,
MAP_REG[nrow][tcol].subreg[19].lly=MAP_REG[nrow][tcol].subreg[6].lly,
   sub4d = 1;
```

```
 if (sub4d == 1)
MAP_REG[nrow][tcol].subreg[19].urx=MAP_REG[nrow][tcol].subreg[19].urx+f,
MAP_REG[nrow][tcol].subreg[19].ury=MAP_REG[nrow][tcol].subreg[19].ury,
MAP_REG[nrow][tcol].subreg[19].llx=MAP_REG[nrow][tcol].subreg[19].llx,
MAP_REG[nrow][tcol].subreg[19].lly=MAP_REG[nrow][tcol].subreg[19].lly;
} /* endif netnum != 0 */


return(0);

}
```

UPDATE5A5B.C

```
#include "globe.h"
#include <stdio.h>

/*****************************************************************/
/*                                                               */
/* ROUTINE TO HANDLE SITUATION WHERE ENTERING                    */
/* NET NUMBER FROM BOTTOM IS NOT EQUAL TO ANY                    */
/* OCCUPIED TRACKS.USED IN BOTH LEFT SCAN AND                    */
/* RIGHT SCAN REGION.                                            */
/*                                                               */
/*****************************************************************/

PROC update5a5b(k,trow,tcol,row,MAX,j,dimension)

int k,trow,tcol,row,MAX,j,dimension;
{
int check,save,tcount,dense,i,srow,ydim,fact,m;
int sub5,sub5a,sub5b,sub5c,sub5d,sub5e,sub5f,sub5g;
int netnum,saveritecol,factor,test;
ydim = dimension;
netnum = 0;
factor = 0;
saveritecol = 0;


netnum = MAP_REG[0][tcol].subreg[1].netnum;
i = tcol + 1;
while(i != col + 1){
if(tkeep[i] == netnum || bkeep[i] == netnum)
    saveritecol = i,
    factor = (saveritecol - tcol)*16 -12;


                /* note here we only have to check */
               /* one way for other pins because  */
              /* this is after initial scan is   */
             /* complete therefore we are       */
            /* scanning from left to right     */


i = i + 1;
}/* end while loop */


for(i = 1;i<=MAX;i++){
test = MAP_REG[i][tcol].subreg[1].netnum;
if(test == netnum && test != 0)
 j = i;
}

check = strcmp(MAP_REG[row][tcol].subreg[1].layer,metal1);
if(check == 0)
  k= 1;

if(check != 0)
```

```
  k= 2;


    nrow = j;
    save = MAX;

if(tcol >= start + 1)
nrow = rowfind(saveritecol,tcol,MAX);









sub5 = 0;
sub5a = 0;
sub5b = 0;
sub5c = 0;
sub5d = 0;
sub5e = 0;
sub5f = 0;
sub5g = 0;




if(netnum != 0 && nrow != 0){
   if (k  == 1)
MAP_REG[nrow][tcol].subreg[1].urx=MAP_REG[row][tcol].subreg[1].urx+9,
MAP_REG[nrow][tcol].subreg[1].ury=MAP_REG[row][tcol].subreg[1].ury,
MAP_REG[nrow][tcol].subreg[1].llx=MAP_REG[row][tcol].subreg[1].llx,
MAP_REG[nrow][tcol].subreg[1].lly=MAP_REG[row][tcol].subreg[1].lly,
   sub5e =1;


 if (k == 2)
MAP_REG[nrow][tcol].subreg[2].urx=MAP_REG[row][tcol].subreg[2].urx+9,
MAP_REG[nrow][tcol].subreg[2].ury=MAP_REG[row][tcol].subreg[2].ury,
MAP_REG[nrow][tcol].subreg[2].llx=MAP_REG[row][tcol].subreg[2].llx,
MAP_REG[nrow][tcol].subreg[2].lly=MAP_REG[row][tcol].subreg[2].lly,
   m = 0,
   sub5a = 1;


   if(sub5e == 1)
MAP_REG[nrow][tcol].subreg[4].urx=MAP_REG[nrow][tcol].subreg[1].urx,
MAP_REG[nrow][tcol].subreg[4].ury=MAP_REG[nrow][tcol].subreg[1].ury,
```

```
MAP_REG[nrow][tcol].subreg[4].llx=MAP_REG[nrow][tcol].subreg[1].urx-4,
MAP_REG[nrow][tcol].subreg[4].lly=MAP_REG[nrow][tcol].subreg[1].lly-1,
   sub5c =1;


   if (sub5c == 1)
MAP_REG[nrow][tcol].subreg[2].urx=MAP_REG[nrow][tcol].subreg[4].urx,
MAP_REG[nrow][tcol].subreg[2].ury=MAP_REG[nrow][tcol].subreg[4].ury+4,
MAP_REG[nrow][tcol].subreg[2].llx=MAP_REG[nrow][tcol].subreg[4].llx+1,
MAP_REG[nrow][tcol].subreg[2].lly=MAP_REG[nrow][tcol].subreg[4].lly+4,
   m = 4,
   sub5a = 1;


 if (sub5a == 1)
   fact = (nrow * ydim) -(ydim - 7) - m,
MAP_REG[nrow][tcol].subreg[5].urx=MAP_REG[nrow][tcol].subreg[2].urx,
MAP_REG[nrow][tcol].subreg[5].ury=MAP_REG[nrow][tcol].subreg[2].ury+fact,
MAP_REG[nrow][tcol].subreg[5].llx=MAP_REG[nrow][tcol].subreg[2].urx-3,
MAP_REG[nrow][tcol].subreg[5].lly=MAP_REG[nrow][tcol].subreg[2].ury,
   sub5b = 1;


 if (sub5b == 1)
MAP_REG[nrow][tcol].subreg[6].urx=MAP_REG[nrow][tcol].subreg[5].urx,
MAP_REG[nrow][tcol].subreg[6].ury=MAP_REG[nrow][tcol].subreg[5].ury+1,
MAP_REG[nrow][tcol].subreg[6].llx=MAP_REG[nrow][tcol].subreg[5].urx-4,
MAP_REG[nrow][tcol].subreg[6].lly=MAP_REG[nrow][tcol].subreg[5].ury-3,
   sub5c = 1;



 if (sub5c == 1 && saveritecol != 0)
MAP_REG[nrow][tcol].subreg[19].urx=
MAP_REG[nrow][tcol].subreg[6].urx+factor,
MAP_REG[nrow][tcol].subreg[19].ury=MAP_REG[nrow][tcol].subreg[6].ury-1,
MAP_REG[nrow][tcol].subreg[19].llx=MAP_REG[nrow][tcol].subreg[6].urx,
MAP_REG[nrow][tcol].subreg[19].lly=MAP_REG[nrow][tcol].subreg[6].lly;



 if (sub5c == 1 && saveritecol == 0)
MAP_REG[nrow][tcol].subreg[19].urx=MAP_REG[nrow][tcol].subreg[6].urx,
MAP_REG[nrow][tcol].subreg[19].ury=MAP_REG[nrow][tcol].subreg[6].ury-1,
MAP_REG[nrow][tcol].subreg[19].llx=MAP_REG[nrow][tcol].subreg[6].urx-12,
MAP_REG[nrow][tcol].subreg[19].lly=MAP_REG[nrow][tcol].subreg[6].lly;

} /* endif netnum != 0 */


 if (sub5c == 1 && saveritecol != 0){
   for(i = tcol;i<= saveritecol;i++){
MAP_REG[nrow][i].subreg[1].netnum = netnum;
}
}
/* update appropriate track with appropriate net# */
```

```
 if (sub5c == 1 && saveritecol == 0){
   MAP_REG[nrow][tcol].subreg[1].netnum = netnum;
}


return(0);

}
```

MESSAGE.C

```
/*****************************************************************************/
/*                                                                           */
/*                                                                           */
/* ROUTINE TO HANDLE SITUATION CHANNEL DENSITY IS                            */
/* EXCEEDED AND MESSAGE IS SENT TO USER TO                                   */
/* INCREASE THE DENSITY.                                                     */
/*                                                                           */
/*****************************************************************************/


#include "globe.h"
#include <stdio.h>
PROC message(L)
int L;
{
int dense;



 dense = MAX + 1;
 printf("THE CHANNEL DENSITY CAPACITY HAS BEEN EXCEEDED !!!\n");
 printf("THEREFORE YOU MUST RESTART THE PROGRAM AND INCREASE\n");
 printf("THE DENSITY BY ONE\n");
 printf("DENSITY = %d",dense);
 printf("    \n");
 printf("START = %d",start);
 printf("    \n");




return(L);
}
```

UPDATER12A.C

```
#include "globe.h"
#include <stdio.h>


PROC updater12a(k,dimension,0,row,tcol)
int k,dimension,0,row,tcol;
{

 /*************************************************************************/
 /*                                                                       */
 /*   ROUTINE TO HANDLE COORDINATE UPDATE FOR                             */
 /*                                                                       */
 /*   RIGHT SCAN REGION ONLY .THIS IS THE SITUATION                       */
 /*                                                                       */
 /*   WHERE ENTERING NETNUM IS METAL1 or METAL2                           */
 /*                                                                       */
 /*   AND THERE IS AN EXISTING TRACK WITH SAME NET NUMBER                 */
 /*                                                                       */
 /*   IN THE ROUTING REGION. ENTRANCE IS FROM TOP OF                      */
 /*                                                                       */
 /*   REGION.                                                             */
 /*************************************************************************/

int j,dim0,dim1,ydim,trow;
int sub1a,sub1b,sub1c,sub1d,save;

dim1 = dimension - 8;
dim0 = dimension - 4;
ydim = dimension;
trow = 0;
save = 0;
trow = row -1;
 sub1a = 0;
 sub1b = 0;
 sub1c = 0;
 sub1d = 0;


 if (k == 1)
MAP_REG[trow][tcol].subreg[3].urx=MAP_REG[row][tcol].subreg[1].llx+4,
MAP_REG[trow][tcol].subreg[3].ury=MAP_REG[row][tcol].subreg[1].lly,
MAP_REG[trow][tcol].subreg[3].llx=MAP_REG[row][tcol].subreg[1].llx,
MAP_REG[trow][tcol].subreg[3].lly=MAP_REG[row][tcol].subreg[1].lly-4,
 sub1a = 1;

 else if (k == 2)
MAP_REG[trow][tcol].subreg[2].urx=MAP_REG[row][tcol].subreg[2].llx+3,
MAP_REG[trow][tcol].subreg[2].ury=MAP_REG[row][tcol].subreg[2].lly,
MAP_REG[trow][tcol].subreg[2].llx=MAP_REG[row][tcol].subreg[2].llx,
MAP_REG[trow][tcol].subreg[2].lly=MAP_REG[row][tcol].subreg[2].lly-dim0,
 sub1b = 1;
```

```
 if (sub1a == 1)
MAP_REG[trow][tcol].subreg[2].urx=MAP_REG[trow][tcol].subreg[3].llx+3,
MAP_REG[trow][tcol].subreg[2].ury=MAP_REG[trow][tcol].subreg[3].lly,
MAP_REG[trow][tcol].subreg[2].llx=MAP_REG[trow][tcol].subreg[3].llx,
MAP_REG[trow][tcol].subreg[2].lly=MAP_REG[trow][tcol].subreg[3].lly-dim1,
 sub1b = 1;

 if (sub1b == 1)
 MAP_REG[trow][tcol].subreg[2].urx=MAP_REG[trow][tcol].subreg[2].urx,
 MAP_REG[trow][tcol].subreg[2].ury=MAP_REG[trow][tcol].subreg[2].ury,
 MAP_REG[trow][tcol].subreg[2].llx=MAP_REG[trow][tcol].subreg[2].llx,
 MAP_REG[trow][tcol].subreg[2].lly =
 MAP_REG[trow][tcol].subreg[2].lly-ydim*0,
 sub1c = 1;


   if (sub1c == 1)
MAP_REG[trow][tcol].subreg[4].urx=MAP_REG[trow][tcol].subreg[2].llx+4,
MAP_REG[trow][tcol].subreg[4].ury=MAP_REG[trow][tcol].subreg[2].lly,
MAP_REG[trow][tcol].subreg[4].llx=MAP_REG[trow][tcol].subreg[2].llx,
MAP_REG[trow][tcol].subreg[4].lly=MAP_REG[trow][tcol].subreg[2].lly-4,
   sub1d = 1;



trow = save;

return(0);
}
```

UPDATER3A3B.C

```
/*****************************************************************/
/*                                                               */
/* ROUTING FOR RIGHT SCAN REGION                                 */
/* ROUTINE TO HANDLE SITUATION WHERE PIN IS METAL1 OR            */
/* METAL2 ENTERING FROM TOP OF ROUTING REGION                    */
/* BUT THERE IS NOT AN EXISTING TRACK WITH THE SAME              */
/* NET NUMBER.                                                   */
/*                                                               */
/*****************************************************************/


#include "globe.h"
#include <stdio.h>
PROC updater3a3b(k,trow,tcol,dimension,O,row,col)
int k,trow,tcol,dimension,O,row,col;
{
int i,savritecol,f,nrow,ydim,dim0,dim1,fact,savecol;
int sub3,sub3a,sub3b,sub3c,sub3d,netnum,limit0,limit1;

   netnum = tkeep[tcol];
   dim0 = dimension -4;
   dim1 = dimension -8;
  ydim = dimension;
        savecol = 0;
         fact = 0;
         savritecol = 0;
         sub3  = 0;
         sub3a = 0;
         sub3b = 0;
         sub3c = 0;
         sub3d = 0;



i = tcol + 1;
while(i <= col){
if(tkeep[i] == netnum || bkeep[i] == netnum)
    savritecol = i,
    f = (savritecol - tcol)*16;

i = i + 1;
}/* end while loop */



i = tcol - 1;
while(i != 0){
if(tkeep[i] == netnum || bkeep[i] == netnum)
    savecol = i,
    fact = (tcol - savecol)*16;

i = i - 1;
}/* end while loop */
```

```
nrow = rowfind1(savecol,savritecol,tcol,MAX);
O = MAX - nrow;

if(netnum != 0 && nrow != 0){
   if (k == 1)
MAP_REG[trow][tcol].subreg[3].urx=MAP_REG[row][tcol].subreg[1].llx+4,
MAP_REG[trow][tcol].subreg[3].ury=MAP_REG[row][tcol].subreg[1].lly,
MAP_REG[trow][tcol].subreg[3].llx=MAP_REG[row][tcol].subreg[1].llx,
MAP_REG[trow][tcol].subreg[3].lly=MAP_REG[row][tcol].subreg[1].lly-4,
   sub3 = 1;

   else if (k == 2)
MAP_REG[trow][tcol].subreg[2].urx=MAP_REG[row][tcol].subreg[2].llx+3,
MAP_REG[trow][tcol].subreg[2].ury=MAP_REG[row][tcol].subreg[2].lly,
MAP_REG[trow][tcol].subreg[2].llx=MAP_REG[row][tcol].subreg[2].llx,
MAP_REG[trow][tcol].subreg[2].lly=MAP_REG[row][tcol].subreg[2].lly-dim0,
   sub3b = 1;



   if (sub3== 1)
MAP_REG[trow][tcol].subreg[2].urx=MAP_REG[trow][tcol].subreg[3].llx+3,
MAP_REG[trow][tcol].subreg[2].ury=MAP_REG[trow][tcol].subreg[3].lly,
MAP_REG[trow][tcol].subreg[2].llx=MAP_REG[trow][tcol].subreg[3].llx,
MAP_REG[trow][tcol].subreg[2].lly=MAP_REG[trow][tcol].subreg[3].lly-dim1,
   sub3b = 1;


   if (sub3b == 1)
MAP_REG[trow][tcol].subreg[2].urx=MAP_REG[trow][tcol].subreg[2].urx,
MAP_REG[trow][tcol].subreg[2].ury=MAP_REG[trow][tcol].subreg[2].ury,
MAP_REG[trow][tcol].subreg[2].llx=MAP_REG[trow][tcol].subreg[2].llx,
MAP_REG[trow][tcol].subreg[2].lly=
MAP_REG[trow][tcol].subreg[2].lly-ydim*O,
   sub3c =1;


   if (sub3c == 1)
MAP_REG[trow][tcol].subreg[4].urx=MAP_REG[trow][tcol].subreg[2].llx+4,
MAP_REG[trow][tcol].subreg[4].ury=MAP_REG[trow][tcol].subreg[2].lly,
MAP_REG[trow][tcol].subreg[4].llx=MAP_REG[trow][tcol].subreg[2].llx,
MAP_REG[trow][tcol].subreg[4].lly=MAP_REG[trow][tcol].subreg[2].lly-4,
   sub3d = 1;


   if (sub3d == 1 )
MAP_REG[trow][tcol].subreg[19].urx=MAP_REG[trow][tcol].subreg[4].llx+12,
MAP_REG[trow][tcol].subreg[19].ury=MAP_REG[trow][tcol].subreg[4].lly+3,
MAP_REG[trow][tcol].subreg[19].llx=MAP_REG[trow][tcol].subreg[4].urx,
MAP_REG[trow][tcol].subreg[19].lly=MAP_REG[trow][tcol].subreg[4].lly;
```

```
   if (savecol != 0)
MAP_REG[trow][tcol].subreg[19].urx =MAP_REG[trow][tcol].subreg[19].urx,
MAP_REG[trow][tcol].subreg[19].ury=MAP_REG[trow][tcol].subreg[19].ury,
MAP_REG[trow][tcol].subreg[19].llx=
MAP_REG[trow][tcol].subreg[19].llx-fact,
MAP_REG[trow][tcol].subreg[19].lly=MAP_REG[trow][tcol].subreg[19].lly;



   if (savritecol != 0)
MAP_REG[trow][tcol].subreg[19].urx=MAP_REG[trow][tcol].subreg[19].urx+f,
MAP_REG[trow][tcol].subreg[19].ury=MAP_REG[trow][tcol].subreg[19].ury,
MAP_REG[trow][tcol].subreg[19].llx=MAP_REG[trow][tcol].subreg[19].llx,
MAP_REG[trow][tcol].subreg[19].lly=MAP_REG[trow][tcol].subreg[19].lly;

}  /* endif netnum != 0 */



if(tcol == col)
   limit1 = 0;
else if(tcol != col)
   limit1 = 1;


if(savritecol == col)
   limit0 = 0;
else if(savritecol != col)
   limit0 = 1;









    /* update track with the appropriate netnumber */



 if (savritecol != 0 && savecol == 0){
for(i = tcol;i<= savritecol+1;i++){
  MAP_REG[nrow][i].subreg[1].netnum = netnum;
}
}


 if (savritecol != 0 && savecol != 0){
for(i = savecol;i<= savritecol+1;i++){
```

```
  MAP_REG[nrow][i].subreg[1].netnum = netnum;
}
}


 if (savritecol == 0 && savecol != 0){
for(i = savecol;i<= tcol+1;i++){
  MAP_REG[nrow][i].subreg[1].netnum = netnum;
}
}



 if (savritecol == 0 && savecol == 0){
   MAP_REG[nrow][tcol].subreg[1].netnum = tkeep[tcol];
}









return(0);
}
```

ROWFIND.C

```
#include "globe.h"
#include <stdio.h>



/*****************************************************************************/
/*                                                                           */
/* ROUTINE TO FIND AN EMPTY TRACK BETWEEN TO                                 */
/* PINS IN THE RIGHT SCAN AREA.                                              */
/* USED AS A CALL FROM UPDATE5A5B.C                                          */
/*                                                                           */
/*                                                                           */
/*****************************************************************************/

PROC rowfind(ritecol,tcol,MAX)

int ritecol,tcol,MAX;
{
int i,j,storecount,nrow;

nrow = 0;



if(ritecol != 0){
for(i = 1;i<= MAX;i++){
 storecount = 0;

for(j =tcol;j<=ritecol;j++){
if(MAP_REG[i][j].subreg[1].netnum == 0)
    storecount = storecount + 1;
if(storecount == ritecol-tcol)
     nrow = i,
     j = ritecol + 1,
     i = MAX + 1;
}
}
}







if(ritecol == 0){
for(i = 1;i<= MAX;i++){
if(MAP_REG[i][tcol].subreg[1].netnum == 0)
     nrow = i,
     i = MAX + 1;
}
}
```

```
if(nrow == 0)
  message(0);


return(nrow);
}
```

ROWFIND1.C

```
#include "globe.h"
#include <stdio.h>

/*****************************************************************/
/*                                                               */
/* ROUTINE TO FIND AN EMPTY TRACK BETWEEN TO                     */
/* PINS IN THE RIGHT SCAN AREA.                                  */
/* USED AS CALL FROM UPDATER3A3B.c                               */
/*                                                               */
/*                                                               */
/*****************************************************************/

PROC rowfind1(savecol,saveritecol,tcol,MAX)

int savecol,saveritecol,tcol,MAX;
{
int i,j,storecount,nrow;

nrow = 0;



if(saveritecol != 0 && savecol !=0){
for(i = 1; i<= MAX; i++){
 storecount = 0;

for(j =savecol; j<=saveritecol; j++){
if(MAP_REG[i][j].subreg[1].netnum == 0)
    storecount = storecount + 1;
if(storecount == saveritecol-savecol)
     nrow = i,
     j = saveritecol + 1,
     i = MAX + 1;
}
}
}



if(saveritecol != 0 && savecol == 0){
for(i = 1; i<= MAX; i++){
 storecount = 0;

for(j =tcol; j<=saveritecol; j++){
if(MAP_REG[i][j].subreg[1].netnum == 0)
    storecount = storecount + 1;
if(storecount == saveritecol-tcol)
     nrow = i,
     j = saveritecol + 1,
     i = MAX + 1;
}
}
}
```

```
if(saveritecol == 0 && savecol != 0){
for(i = 1;i<= MAX;i++){
 storecount = 0;

for(j =savecol;j<=tcol;j++){
if(MAP_REG[i][j].subreg[1].netnum == 0)
    storecount = storecount + 1;
if(storecount == tcol - savecol)
     nrow = i,
     j = tcol + 1,
     i = MAX + 1;
}
}
}
```

```
if(saveritecol == 0 && savecol == 0 ){
for(i = 1;i<= MAX;i++){
if(MAP_REG[i][tcol].subreg[1].netnum == 0)
     nrow = i,
     i = MAX + 1;
}
}
```

```
if(nrow == 0)
  message(0);


return(nrow);
}
```

NUMBER.C

```
#include "globe.h"
#include <stdio.h>

/*****************************************************************************/
/*                                                                           */
/*                                                                           */
/* THIS ROUTINE CALCULATES THE MAXIMUM                                       */
/* TRACKS FOR ANY GIVEN COLUMN IN THE                                        */
/* ROUTING AREA AND CONSIDERS THE                                            */
/* SAME NET ONLY ONCE PER COLUMN PER SIDE                                    */
/*                                                                           */
/*                                                                           */
/*****************************************************************************/



PROC number(col,top,bottom,limit)
int col,top[],bottom[],limit;
{
int *psave,comp(),left[101],right[101];
int j,i,p,track;
for(j=0;j<col;j++){
start=j+1;
save[j] = *(top+j);       /* read in the top terminal nets */
stop = (2*j)+2;           /* on the left side of column    */
}
i=0;
for(j=start;j<stop;j++){  /* do the same for bottom */
save[j]= *(bottom+i);
i=i+1;
}

/* sort the "left" nets in ascending order */

qsort(save,stop,sizeof(int),comp);


for (i=0;i<101;i++){
left[i]=0;
}
i= 0;
psave = &save[0];
left[0] = *psave;
while (i<stop-1){
i=i+1;
if (save[i] != save[i+1])            /* get rid of duplicate nets */
left[i] = *(psave +i+1);
}



/* now do the same for right side as was done for the left side */
```

```
i=0;
for(j=col;j<=limit;j++){
start=i+1;
save[i] = *(top+j);
stop = (2*i)+2;
i=i+1;
}
i=col;
for(j=start;j<stop;j++){
save[j]= *(bottom+i);
i=i+1;
}


qsort(save,stop,sizeof(int),comp);



for (i=0;i<101;i++){
right[i]=0;
}
i= 0;
psave = &save[0];
right[0] = *psave;
while (i<stop-1){
i=i+1;
if (save[i] != save[i+1])
right[i] = *(psave +i+1);
}



track=0;


   /* now correlate data and see how many matches */
  /* there are and that is the number of tracks */
 /* for the current column                      */

for(i=0;i<stop-1;i++)
   for(j=0;j<stop{1;j++){
if(left[i]==0)
  track=track;
else if(left[i] == right[j])
  track=track+1;
else if(left[i] != right[j])
  track=track;
}
```

```
return(track);                          /* return the value */

}


comp(i,j)
int *i,*j;
{
return *i-*j;
}
```

INPUT.C

```
#include "globe.h"
#include <stdio.h>
```

```
                        /*****************************************************/
                        /* THIS ROUTINE DOES THE CHANNEL                     */
                        /*                                                   */
                        /* INTIALIZATION FOR NEW INPUTS                      */
PROC input(q)           /*                                                   */
                        /* AND THEN STORES THE DATA IN A FILE                */
                        /*                                                   */
                        /* CALLED TEMP.DATA FOR FUTURE USE.                  */
                        /*****************************************************/

int q;
{
FILE *fp,*fopen();
int n;                  /* user specified number of terminals   */
int i,limit,j;
int rbound;              /* right bound of switchbox*/
int count;              /* counter used as situation dictates    */
int chck,check,*ptop,*pbot,k;
```

```
/*************************************************************************/
/*                                                                       */
/*                                                                       */
/*                       USER INPUT SECTION                              */
/*                                                                       */
/*                                                                       */
/*                                                                       */
/*************************************************************************/
```

```
fp = fopen("temp.data","w");

/* USER SPECIFIES THE NUMBER OF TERMINALS                                */

printf(" \n");
printf(" \n");
printf("specify the number of terminals n =  ");
scanf("%3d",&n);
printf(" \n");
fprintf(fp,"%d\n",n);
printf("THE NUMBER OF TERMINALS IS%3d\n", n);
```

```
/* initialize limit for netlist
   and row and col                                          */

for(i=1;i<2;i++){
limit=n+1;
row = n-1;
col = n+1;
rbound=col+1;
count=1;
}

top[0]=0;
bottom[0]=0;
top[rbound]=0;
bottom[rbound]=0;
fprintf(fp,"%d\n",top[0]);
fprintf(fp,"%d\n",bottom[0]);
fprintf(fp,"%d\n",top[rbound]);
fprintf(fp,"%d\n",bottom[rbound]);



/* USER INPUTS A SPECIFIED NETLIST                          */

for (i=1;i<limit;i++){
top[0]=0;
bottom[0]=0;
top[rbound]=0;
bottom[rbound]=0;
printf("   \n");
printf("ENTER THE NETLIST # FOR TOP%3d",i," = ");
printf("=");
scanf("%2d",&top[i]);
fprintf(fp,"%d\n",top[i]);
printf("   \n");
printf("ENTER THE NETLIST # FOR BOTTOM%3d",i," = ");
printf("=");
scanf("%3d",&bottom[i]);
fprintf(fp,"%d\n",bottom[i]);
    }
Tracks[0]=0;
printf("TRACKS PER COLUMN IS AS FOLLOWS\n");
printf(" \n");
for (i=1;i<=col;i++){

/* call number's and find */
/* max tracks for that column */

Tracks[i] = number(i,&top[0],&bottom[0],limit);
scol[i] = Tracks[i];                    /* save for later use */
printf("tracks = %d\n",Tracks[i]);
if (Tracks[i]<=Tracks[i-1])
  Tracks[i] = Tracks[i-1];
else if (Tracks[i]>Tracks[i-1])
```

```
    MAX=Tracks[i];
}
fprintf(fp,"%d\n",MAX);
printf("                \n");
printf("THE CHANNEL DENSITY IS THE MAXIMUM");
printf("                \n");
printf("HORIZONTAL TRACKS = %3d\n",MAX);
printf("            \n");



/*****************************************************************************/
/*                                                                           */
/*            NOW INPUT NETLAYER FOR TOP PINS                                */
/*                                                                           */
/*****************************************************************************/

for (i=1;i< col;i++){
row = MAX + 1;
tcol = i;
printf("enter the netlayer for top%d:",i);

scanf("%s",response);
metal1 = "metal1";
check = strcmp(response,metal1);
fprintf(fp,"%d\n",check);
chck = check;
if(check == 0)
MAP_REG[row][tcol].subreg[1].layer = "metal1",
MAP_REG[row][tcol].subreg[7].layer = "metal1",
fprintf(fp,"%s\n",MAP_REG[row][tcol].subreg[1].layer),
fprintf(fp,"%s\n",MAP_REG[row][tcol].subreg[7].layer);
 else if (check != 0)
MAP_REG[row][tcol].subreg[2].layer = "metal2",
MAP_REG[row][tcol].subreg[7].layer = "metal2",

fprintf(fp,"%s\n",MAP_REG[row][tcol].subreg[2].layer),
fprintf(fp,"%s\n",MAP_REG[row][tcol].subreg[7].layer);

if(chck == 0)
   k=1;
else if(chck != 0)
  k=2;




/*****************************************************************************/
/*                                                                           */
/*    INPUTTING NET NAME & NUMBER FOR TOP PINS                               */
/*                                                                           */
/*****************************************************************************/
```

```
printf("TYPE IN top%d:",i);
scanf("%s",MAP_REG[row][tcol].subreg[k].netname);
ptop = &top[i];
MAP_REG[row][tcol].subreg[1].netnum = *ptop;
MAP_REG[row][tcol].subreg[7].netnum = *ptop;

/*****************************************************************************/
/* FOR TOP PINS                                                              */
/* SET THE COORDINATES SO THAT ALL ALIGNMENT IS ON LEFT                      */
/* BOUNDARY OF GRID OR BOTTOM BOUNDARY OF GRID                               */
/*                                                                           */
/*****************************************************************************/


MAP_REG[row][tcol].subreg[k].llx = (tcol - 1) * 24;
MAP_REG[row][tcol].subreg[k].lly = MAX*24;
MAP_REG[row][tcol].subreg[k].urx = (tcol - 1 ) * 24 + 3;
MAP_REG[row][tcol].subreg[k].ury = MAX*24 + 3;

fprintf(fp,"%d\n",MAP_REG[row][tcol].subreg[k].llx);
fprintf(fp,"%d\n",MAP_REG[row][tcol].subreg[k].lly);
fprintf(fp,"%d\n",MAP_REG[row][tcol].subreg[k].urx);
fprintf(fp,"%d\n",MAP_REG[row][tcol].subreg[k].ury);

MAP_REG[row][tcol].subreg[7].llx = (tcol - 1) * 24;
MAP_REG[row][tcol].subreg[7].lly = MAX*24;
MAP_REG[row][tcol].subreg[7].urx = MAP_REG[row][tcol].subreg[7].llx;
MAP_REG[row][tcol].subreg[7].ury = MAP_REG[row][tcol].subreg[7].lly;

fprintf(fp,"%d\n",MAP_REG[row][tcol].subreg[7].llx);
fprintf(fp,"%d\n",MAP_REG[row][tcol].subreg[7].lly);
fprintf(fp,"%d\n",MAP_REG[row][tcol].subreg[7].urx);
fprintf(fp,"%d\n",MAP_REG[row][tcol].subreg[7].ury);

printf("\n")



}



/*****************************************************************************/
/*                                                                           */
/*            NOW INPUT NETLAYER FOR BOTTOM PINS                             */
/*                                                                           */
/*****************************************************************************/
for (i=1;i< col;i++){
row = 0;
tcol = i;
printf("enter the netlayer for bottom%d:",i);

scanf("%s",response);
metal1 = "metal1";
check = strcmp(response,metal1);
```

```
fprintf(fp,"%d\n",check);
chck = check;
if(check == 0)
MAP_REG[row][tcol].subreg[1].layer = "metal1",
MAP_REG[row][tcol].subreg[7].layer = "metal1",
fprintf(fp,"%s\n",MAP_REG[row][tcol].subreg[1].layer),
fprintf(fp,"%s\n",MAP_REG[row][tcol].subreg[7].layer);

 else if (check != 0)
MAP_REG[row][tcol].subreg[2].layer = "metal2",
MAP_REG[row][tcol].subreg[7].layer = "metal2",
fprintf(fp,"%s\n",MAP_REG[row][tcol].subreg[2].layer),
fprintf(fp,"%s\n",MAP_REG[row][tcol].subreg[7].layer);

if(chck == 0)
   k=1;
else if(chck != 0)
  k=2;

/*****************************************************************************************/
/*                                                                                       */
/*    INPUTTING NET NAME & NUMBER FOR BOTTOM PINS                                        */
/*                                                                                       */
/*****************************************************************************************/


printf("TYPE IN bot%d:",i);
scanf("%s",MAP_REG[row][tcol].subreg[k].netname);
pbot = &bottom[i];
MAP_REG[row][tcol].subreg[1].netnum = *pbot;
MAP_REG[row][tcol].subreg[7].netnum = *pbot;

/*****************************************************************************************/
/* FOR BOTTOM PINS                                                                       */
/* SET THE COORDINATES SO THAT ALL ALIGNMENT IS ON LEFT                                  */
/* BOUNDARY OF GRID OR BOTTOM BOUNDARY OF GRID                                           */
/*                                                                                       */
/*****************************************************************************************/


MAP_REG[row][tcol].subreg[k].llx = (tcol - 1) * 24;
MAP_REG[row][tcol].subreg[k].lly = -7;
MAP_REG[row][tcol].subreg[k].urx = (tcol - 1 ) * 24 + 3;
MAP_REG[row][tcol].subreg[k].ury = -4;


fprintf(fp,"%d\n",MAP_REG[row][tcol].subreg[k].llx);
fprintf(fp,"%d\n",MAP_REG[row][tcol].subreg[k].lly);
fprintf(fp,"%d\n",MAP_REG[row][tcol].subreg[k].urx);
fprintf(fp,"%d\n",MAP_REG[row][tcol].subreg[k].ury);

MAP_REG[row][tcol].subreg[7].llx = (tcol - 1) * 24;
MAP_REG[row][tcol].subreg[7].lly = -7;
MAP_REG[row][tcol].subreg[7].urx = MAP_REG[row][tcol].subreg[7].llx;
MAP_REG[row][tcol].subreg[7].ury = MAP_REG[row][tcol].subreg[7].lly;
```

```
fprintf(fp,"%d\n",MAP_REG[row][tcol].subreg[7].llx);
fprintf(fp,"%d\n",MAP_REG[row][tcol].subreg[7].lly);
fprintf(fp,"%d\n",MAP_REG[row][tcol].subreg[7].urx);
fprintf(fp,"%d\n",MAP_REG[row][tcol].subreg[7].ury);

printf("\n");

}

fclose(fp);

return(0);
}
```

INITIALIZE.C

```
#include "globe.h"
#include <stdio.h>
                        /*********************************************** */
                        /* THIS ROUTINE DOES THE CHANNEL                */
                        /*                                              */
                        /* INTIALIZATION FOR NEW INPUTS                 */
                        /* AND THEN STORES THE DATA IN A FILE           */
                        /*                                              */
main()                  /* CALLED TEMP.DATA FOR FUTURE USE.             */
{                       /***********************************************/

int n;                 /* user specified number of terminals   */
int i,limit,j;
int rbound;             /* right bound of switchbox*/
int count;             /* counter used as situation dictates    */
int chck,check,*ptop,*pbot,k;
FILE *fp,*fopen();


/*****************************************************************************/
/*                                                                           */
/*                                                                           */
/*                       USER INPUT SECTION                                  */
/*                                                                           */
/*                                                                           */
/*                                                                           */
/*****************************************************************************/



fp = fopen("temp.data","w");


/* USER SPECIFIES THE NUMBER OF TERMINALS                                    */
printf(" \n");
printf(" \n");
printf("THIS PROGRAM ALLOWS THE USER TO CREATE A DATA FILE \n");
printf("FOR THE NPGS ROUTER                                \n ");
printf("IT WILL BE STORED IN A FILE CALLED TEMP.DATA       \n ");



printf(" \n");
printf(" \n");
printf("specify the number of terminals n =  ");
scanf("%3d",&n);
printf(" \n");
fprintf(fp,"%d\n",n);
printf("THE NUMBER OF TERMINALS IS%3d\n", n);

/* initialize limit for netlist
```

```
   and row and col                                                    */

for(i=1;i<2;i++){
limit=n+1;
row = n-1;
col = n+1;
rbound=col+1;
count=1;
}

top[0]=0;
bottom[0]=0;
top[rbound]=0;
bottom[rbound]=0;
fprintf(fp,"%d\n",top[0]);
fprintf(fp,"%d\n",bottom[0]);
fprintf(fp,"%d\n",top[rbound]);
fprintf(fp,"%d\n",bottom[rbound]);



/* USER INPUTS A SPECIFIED NETLIST                                   */

for (i=1;i<limit;i++){
top[0]=0;
bottom[0]=0;
top[rbound]=0;
bottom[rbound]=0;
printf("   \n");
printf("ENTER THE NETLIST # FOR TOP%3d",i," = ");
printf("=");
scanf("%2d",&top[i]);
fprintf(fp,"%d\n",top[i]);
printf("   \n");
printf("ENTER THE NETLIST # FOR BOTTOM%3d",i," = ");
printf("=");
scanf("%3d",&bottom[i]);
fprintf(fp,"%d\n",bottom[i]);
    }
Tracks[0]=0;
printf("TRACKS PER COLUMN IS AS FOLLOWS\n");
printf(" \n");
for (i=1;i<=col;i++){
Tracks[i] = number(i,&top[0],&bottom[0],limit);
/* call number's and find */
/* max tracks for that column */

scol[i] = Tracks[i];                    /* save for later use */
printf("tracks = %d\n",Tracks[i]);
if (Tracks[i]<=Tracks[i-1])
  Tracks[i] = Tracks[i-1];
else if (Tracks[i]>Tracks[i-1])
    MAX=Tracks[i];
}
fprintf(fp,"%d\n",MAX);
printf("               \n");
```

```
printf("THE CHANNEL DENSITY IS THE MAXIMUM");
printf("                \n");
printf("HORIZONTAL TRACKS = %3d\n",MAX);
printf("            \n");



/*****************************************************************************/
/*                                                                           */
/*             NOW INPUT NETLAYER FOR TOP PINS                               */
/*                                                                           */
/*****************************************************************************/

for (i=1;i< col;i++){
row = MAX + 1;
tcol = i;
printf("enter the netlayer for top%d:",i);

scanf("%s",response);
metal1 = "metal1";
check = strcmp(response,metal1);
fprintf(fp,"%d\n",check);
chck = check;
if(check == 0)
MAP_REG[row][tcol].subreg[1].layer = "metal1",
MAP_REG[row][tcol].subreg[7].layer = "metal1",
fprintf(fp,"%s\n",MAP_REG[row][tcol].subreg[1].layer),
fprintf(fp,"%s\n",MAP_REG[row][tcol].subreg[7].layer);
 else if (check != 0)
MAP_REG[row][tcol].subreg[2].layer = "metal2",
MAP_REG[row][tcol].subreg[7].layer = "metal2",

fprintf(fp,"%s\n",MAP_REG[row][tcol].subreg[2].layer),
fprintf(fp,"%s\n",MAP_REG[row][tcol].subreg[7].layer);

if(chck == 0)
   k=1;
else if(chck != 0)
  k=2;


/*****************************************************************************/
/*                                                                           */
/*    INPUTTING NET NAME & NUMBER FOR TOP PINS                               */
/*                                                                           */
/*****************************************************************************/

printf("TYPE IN top%d:",i);
scanf("%s",MAP_REG[row][tcol].subreg[k].netname);
ptop = &top[i];
MAP_REG[row][tcol].subreg[1].netnum = *ptop;
MAP_REG[row][tcol].subreg[7].netnum = *ptop;
```

```
/*****************************************************************************/
/* FOR TOP PINS                                                              */
/* SET THE COORDINATES SO THAT ALL ALIGNMENT IS ON LEFT                      */
/* BOUNDARY OF GRID OR BOTTOM BOUNDARY OF GRID                               */
/*                                                                           */
/*****************************************************************************/


MAP_REG[row][tcol].subreg[k].llx = (tcol - 1) * 24;
MAP_REG[row][tcol].subreg[k].lly = MAX*24;
MAP_REG[row][tcol].subreg[k].urx = (tcol - 1 ) * 24 + 3;
MAP_REG[row][tcol].subreg[k].ury = MAX*24 + 3;

fprintf(fp,"%d\n",MAP_REG[row][tcol].subreg[k].llx);
fprintf(fp,"%d\n",MAP_REG[row][tcol].subreg[k].lly);
fprintf(fp,"%d\n",MAP_REG[row][tcol].subreg[k].urx);
fprintf(fp,"%d\n",MAP_REG[row][tcol].subreg[k].ury);

MAP_REG[row][tcol].subreg[7].llx = (tcol - 1) * 24;
MAP_REG[row][tcol].subreg[7].lly = MAX*24;
MAP_REG[row][tcol].subreg[7].urx = MAP_REG[row][tcol].subreg[7].llx;
MAP_REG[row][tcol].subreg[7].ury = MAP_REG[row][tcol].subreg[7].lly;

fprintf(fp,"%d\n",MAP_REG[row][tcol].subreg[7].llx);
fprintf(fp,"%d\n",MAP_REG[row][tcol].subreg[7].lly);
fprintf(fp,"%d\n",MAP_REG[row][tcol].subreg[7].urx);
fprintf(fp,"%d\n",MAP_REG[row][tcol].subreg[7].ury);

printf("\n");




}



/*****************************************************************************/
/*                                                                           */
/*             NOW INPUT NETLAYER FOR BOTTOM PINS                            */
/*                                                                           */
/*****************************************************************************/
for (i=1;i< col;i++){
row = 0;
tcol = i;
printf("enter the netlayer for bottom%d:",i);

scanf("%s",response);
metal1 = "metal1";
check = strcmp(response,metal1);
fprintf(fp,"%d\n",check);
chck = check;
if(check == 0)
MAP_REG[row][tcol].subreg[1].layer = "metal1",
MAP_REG[row][tcol].subreg[7].layer = "metal1",
```

```
fprintf(fp,"%s\n",MAP_REG[row][tcol].subreg[1].layer),
fprintf(fp,"%s\n",MAP_REG[row][tcol].subreg[7].layer);

 else if (check != 0)
MAP_REG[row][tcol].subreg[2].layer = "metal2",
MAP_REG[row][tcol].subreg[7].layer = "metal2",
fprintf(fp,"%s\n",MAP_REG[row][tcol].subreg[2].layer),
fprintf(fp,"%s\n",MAP_REG[row][tcol].subreg[7].layer);

if(chck == 0)
   k=1;
else if(chck != 0)
  k=2;

/*****************************************************************************/
/*                                                                           */
/*    INPUTTING NET NAME & NUMBER FOR BOTTOM PINS                            */
/*                                                                           */
/*****************************************************************************/


printf("TYPE IN bot%d:",i);
scanf("%s",MAP_REG[row][tcol].subreg[k].netname);
pbot = &bottom[i];
MAP_REG[row][tcol].subreg[1].netnum = *pbot;
MAP_REG[row][tcol].subreg[7].netnum = *pbot;

/*****************************************************************************/
/* FOR BOTTOM PINS                                                           */
/* SET THE COORDINATES SO THAT ALL ALIGNMENT IS ON LEFT                      */
/* BOUNDARY OF GRID OR BOTTOM BOUNDARY OF GRID                               */
/*                                                                           */
/*****************************************************************************/


MAP_REG[row][tcol].subreg[k].llx = (tcol - 1) * 24;
MAP_REG[row][tcol].subreg[k].lly = -7;
MAP_REG[row][tcol].subreg[k].urx = (tcol - 1 ) * 24 + 3;
MAP_REG[row][tcol].subreg[k].ury = -4;


fprintf(fp,"%d\n",MAP_REG[row][tcol].subreg[k].llx);
fprintf(fp,"%d\n",MAP_REG[row][tcol].subreg[k].lly);
fprintf(fp,"%d\n",MAP_REG[row][tcol].subreg[k].urx);
fprintf(fp,"%d\n",MAP_REG[row][tcol].subreg[k].ury);

MAP_REG[row][tcol].subreg[7].llx = (tcol - 1) * 24;
MAP_REG[row][tcol].subreg[7].lly = -7;
MAP_REG[row][tcol].subreg[7].urx = MAP_REG[row][tcol].subreg[7].llx;
MAP_REG[row][tcol].subreg[7].ury = MAP_REG[row][tcol].subreg[7].lly;

fprintf(fp,"%d\n",MAP_REG[row][tcol].subreg[7].llx);
fprintf(fp,"%d\n",MAP_REG[row][tcol].subreg[7].lly);
fprintf(fp,"%d\n",MAP_REG[row][tcol].subreg[7].urx);
fprintf(fp,"%d\n",MAP_REG[row][tcol].subreg[7].ury);
```

```
printf("\n");

}

fclose(fp);

}
```

```
/*****************************************************************************/
/*                                                                           */
/*                                                                           */
/* THIS ROUTINE CALCULATES THE MAXIMUM TRACKS                                */
/* FOR ANY GIVEN COLUMN IN THE                                               */
/* ROUTING AREA AND CONSIDERS                                                */
/* THE SAME NET ONLY ONCE PER COLUMN PER SIDE                                */
/*                                                                           */
/*                                                                           */
/*                                                                           */
/*****************************************************************************/
```

```
number(col,top,bottom,limit)
int col,top[],bottom[],limit;
{
int *psave,comp(),left[101],right[101];
int j,i,p,track;
for(j=0;j<col;j++){
start=j+1;
save[j] = *(top+j);       /* read in the top terminal nets */
stop = (2*j)+2;           /* on the left side of column    */
}
i=0;
for(j=start;j<stop;j++){  /* do the same for bottom */
save[j]= *(bottom+i);
i=i+1;
}

qsort(save,stop,sizeof(int),comp);
/* sort the "left" nets in ascending order */


for (i=0;i<101;i++){
left[i]=0;
}
i= 0;
psave = &save[0];
left[0] = *psave;
while (i<stop-1){
i=i+1;
```

```
if (save[ i] != save[ i+1] )            /* get rid of duplicate nets */
left[ i] = *(psave +i+1);
}



/* now do the same for right side as was done for the left side */

i=0;
for( j=col; j<=limit; j++){
start=i+1;
save[ i] = *(top+j);
stop = (2*i)+2;
i=i+1;
}
i=col;
for( j=start; j<stop; j++){
save[ j]= *(bottom+i);
i=i+1;
}

qsort(save,stop,sizeof(int),comp);



for (i=0; i<101; i++){
right[ i]=0;
}
i= 0;
psave = &save[ 0];
right[ 0] = *psave;
while (i<stop-1){
i=i+1;
if (save[ i] != save[ i+1] )
right[ i] = *(psave +i+1);
}



track=0;



for( i=0; i<stop-1; i++)
   for( j=0; j<stop-1; j++){
   /* now correlate data and see how many matches */
  /* there are and that is the number of tracks */
if( left[ i]==0)
  track=track;
   /* for the current column                */
else if( left[ i] == right[ j] )
  track=track+1;
else if( left[ i] != right[ j] )
```

```
  track=track;
}
```

```
return(track);                          /* return the value */

}
```

```
comp(i,j)
int *i,*j;
{
return *i-*j;
}
```

GLOBE.H

```
/*****************************************************************************/
/*                                                                           */
/*                                                                           */
/* THESE ARE DEFINITIONS USED IN THE INPUT SECTION                           */
/*      OF THE ROUTING PROCESS IN THE MAIN CALLING                           */
/*   ROUTINE                                                                 */
/*                                                                           */
/*****************************************************************************/

#define PROC

int save[2000];          /* used for inputting data files                    */
int istore[400];         /* used as temporary storage                        */
int top[100];            /* used as storage for top pin nets                 */
int bottom[100];         /* used as storage for bottom pin nets              */
int tkeep[500];          /* used as temporary storage for top nets           */
int bkeep[500];          /* used as temporary storage for bottom nets        */
int scol[100];           /* used in finding starting column                  */
int store[100];          /* used to keep count of cols that were routed      */
int Tracks[101];         /* used to find the MAX density                     */
int MAX;                 /* the maximum density                              */
int row;                 /* the row number                                   */
int trow;                /* temporary row number                             */
int nrow;                /* the number row for offset and painting           */
int num;                 /* number in input data file                        */
int col;                 /* the number of columns                            */
int tcol;                /* temporary col numbers                            */
int stop;                /* stop indicator for incrementing                  */
int start;               /* the start scanning position                      */
int check1;              /* checking values with other values                */
int anscheck;            /* checking answer from user interaction            */

char fname[128];         /* name of data file for router                     */
char *metal1;            /* define metal1                                    */
char *metal2;            /* define metal2                                    */
char buf[256];           /* buffer size for reading in data file             */
char *yes;               /* define yes                                       */
char answer[100];        /* answer from user                                 */
char response[10];       /* response from user                               */
```

```
/*****************************************************************/
/*                                                               */
/*                                                               */
/* LISTED BELOW IS ONE OF THE MAIN STRUCTURES THIS               */
/* PROGRAM WILL UTILIZE. IT SETS UP A MAP REGION IN              */
/* WHICH THE ROUTING IS PERFORMED.                               */
/*                                                               */
/*                                                               */
/*                                                               */
/*                                                               */
/*                                                               */
/*****************************************************************/
struct {
    struct {
char *layer;
char netname[100];
int netnum;
int llx;
int lly;
int urx;
int ury;
}subreg[25];
}MAP_REG [50][50];
```

```
/*****************************************************************/
/*                                                               */
/*                                                               */
/*          subregion1 = metal1                                  */
/*          subregion2 = metal2 left half                        */
/*          subregion3 = via upper left grid                     */
/*          subregion4 = via lower left grid                     */
/*          subregion5 = metal2 right half                       */
/*          subregion6 = via upper right grid                    */
/*          subregion7 = label coordinates                       */
/*          subregion8 = via for intercolumn                     */
/*                       connection(left)                        */
/*          subregion 19 =  metal1 track                         */
/*                          runner                               */
/*                                                               */
/*****************************************************************/
```

```
struct rlist{
  char ltr[256];     /* structure for reading data files */
}list[2000];
```

MAKEFILE

```
#makefile for program ROUTE

OBJECTS = arouter.o in.o out.o  update12a.o update3a3b.o update4a4b.o
          update5a5b.c message.o updater12a.o updater3a3b.o rowfind.o
          number.o rowfind1.o input.o

SOURCES = arouter.c in.c out.c  update12a.c update3a3b.c update4a4b.c
          update5a5b.c message.c updater12a.c updater3a3b.c rowfind.c
          number.c rowfind1.c input.c

route:    $(OBJECTS)

 cc $(OBJECTS) -o route  (check this line)

$(OBJECTS):  globe.h

#end makefile
```

LIST OF REFERENCES

1. Mukherjee, A., *Introduction to nMOS & CMOS VLSI Systems Design*, Prentice-Hall, Englewood Cliffs, NJ, 1986.

2. Ho, W.P.C., and Yun, D.Y.Y., *Planning Strategies For Switchbox Routing*, IEEE Transactions On CAD, Vol. CAD-2, No.4.J., 1985, pp. 463-466.

3. Lee, C., *An Algorithm For Path Connections and Its Applications*, IRE Transactions On Electronic Computers, September 1961, pp. 346-365.

4. Rivest, R.L., and Fiduccia, C.M., *A "Greedy" Channel Router*, Proceedings Nineteenth Design Automation Conference, 1982, pp.418-423.

5. Aho, A.V., Hopcroft, J.E., and Ullman, J.D., *The Design and Analysis of Computer Algorithms*, Addison Wesley, Reading, MA, 1974.

6. Hamachi, G.T., and Ousterhout, J.K., *A Switchbox Router With Obstacle Avoidance*, Twenty First Design Automation Conference, 1984, pp. 173-179.

7. Luk, W.K., *A Greedy Switch-box Router*, Integration, The VLSI Journal 3, 1985, pp. 129-149.

8. Kernighan, B.W.,Schweikert, D.G., and Persky,G., *An Optimum Channel Routing Algorithm for Polycell Layouts of Integrated Circuits*, Proceedings Tenth Design Automation Workshop, 1973, pp. 50-59.

9. Shin, H., and Sangiovanni-Vincentelli, A., *Mighty: A Rip-Up and Re-route Detailed Router*, Proceedings 1986 International Conference On CAD, Santa Clara, CA, November 1986, pp. 2-5.

10. Joobbani, R., *An Artificial Intelligence Approach to VLSI Routing*, Kluwer Academic Publishers, 1986.

11. Cohoon, J.R., and Heck, P.L., *Beaver: A Switchbox Router*, Proceedings IEEE International Conference On Computer Design, October 1987, pp. 336-340.

12. Ousterhout, J. K., *1986 VLSI TOOLS: Still More Works by The Original Artists*, Report # UCB CSD 86 272, Computer Science Division (EECS), University of California, Berkeley, December 1985.

13. Bentley, J.L., and Ottmann, T., *Algorithms for Reporting and Counting Geometric Intersections*, IEEE Transactions On Computers C-28, 9, September, 1979, pp. 643-647.

14. Burstein, M., and Pelavin, R., *Hierarchial Wire Routing*, IEEE Transactions On CAD, Vol. CAD-2, No.4, October 1983, pp. 223-234.

15. Antognetti, P., DeMicheli, G., and Sangiovanni-Vincentelli, A., *Design Systems For VLSI Circuits*, Martinus Nijhoff Publishers, Dordrecht, The Netherlands, July 1986.

16. Du, H. C., and Enbody, R. J., *General Purpose Router*, 24th Design Automation Conference, 1987, pp. 637-640.

17. Silicon Compiler Systems Corporation, *Genesil System Users Manual*, San Jose, CA, February 1988.

18. Eshraghian, K., and Weste, N., *Principles of CMOS VLSI DESIGN A Systems Perspective*, Addison-Wesley Publishing Company, 1985.

19. Northwest LIS, *VLSI Design Tools Reference Manual*, February, 1987.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Chairman, Code 62 Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5000	1
4.	Commandant of the Marine Corps Code TE 06 Headquarters, U.S. Marine Corps Washington, D.C. 20380-0001	1
5.	Dr. C. Yang, Code 62YA Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5000	6
6.	Dr. C. H. Lee, Code 62LE Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5000	1
7.	Dr. H. H. Loomis, Code 62LM Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5000	2
8.	Lt. Emmanouil Zagourakis SMC# 2738 Naval Postgraduate School Monterey, CA 93943-5000	1
9.	CAPT. Michael J. Roderick 56 Gaywood Street North Dartmouth, MA 02747	2

Thesis
R66575 Roderick
c.1 Channel and switchbox routing using a greedy based channel algorithm with outward scanning technique.

30 NOV 87

80487

Thesis
R66575 Roderick
c.1 Channel and switchbox routing using a greedy based channel algorithm with outward scanning technique.

AMERICAN BINDERY
TOPEKA KANSAS

thesR66575
Channel and switchbox routing using a gr
3 2768 000 81488 3
DUDLEY KNOX LIBRARY